MSDS 694
Distributed Computing

DIANE WOODBRIDGE, PH.D

Announcement

Week 5 Lesson : Flipped Class

- Watch the video and follow the directions by **November 27th**.
- I will keep the morning class time (no afternoon) as an office hour.

Assignments

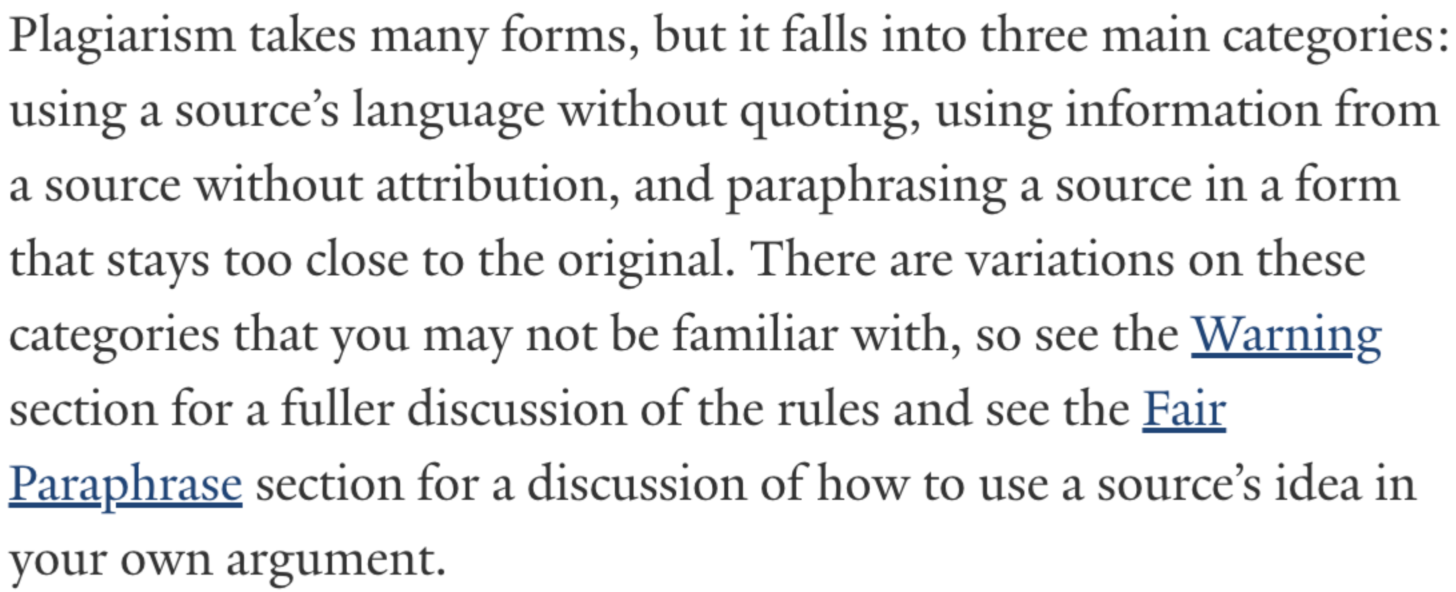
- Group Project Task 2 (Upload data on GCP Storage) - **November 28th**
  - Instructions and videos are available on Canvas.
- Individual Assignment 3 - **Dec 3rd** (Will release early next week)

Quiz 2 - **Dec 2nd 9 am - 10am**

Office Hour : Tuesday 9:15-10 am (**No office hour on Nov 22nd**)

Plagiarism

Many received 0 for plagiarism on their Group Assignment 1.

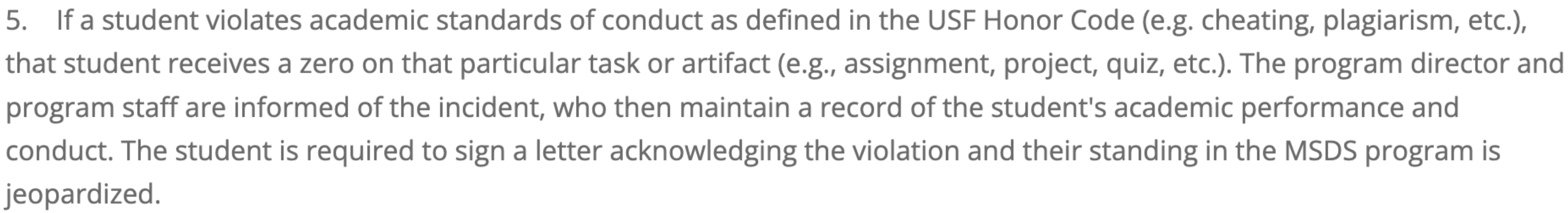


Plagiarism takes many forms, but it falls into three main categories: using a source's language without quoting, using information from a source without attribution, and paraphrasing a source in a form that stays too close to the original. There are variations on these categories that you may not be familiar with, so see the Warning section for a fuller discussion of the rules and see the Fair Paraphrase section for a discussion of how to use a source's idea in your own argument.

https://poorvucenter.yale.edu/undergraduates/using-sources/understanding-and-avoiding-plagiarism/what-plagiarism

Plagiarism

Many received 0 for plagiarism on their Group Assignment 1.



5. If a student violates academic standards of conduct as defined in the USF Honor Code (e.g. cheating, plagiarism, etc.), that student receives a zero on that particular task or artifact (e.g., assignment, project, quiz, etc.). The program director and program staff are informed of the incident, who then maintain a record of the student's academic performance and conduct. The student is required to sign a letter acknowledging the violation and their standing in the MSDS program is jeopardized.

https://myusf.usfca.edu/arts-sciences/data-science/program-policies

Professionalism

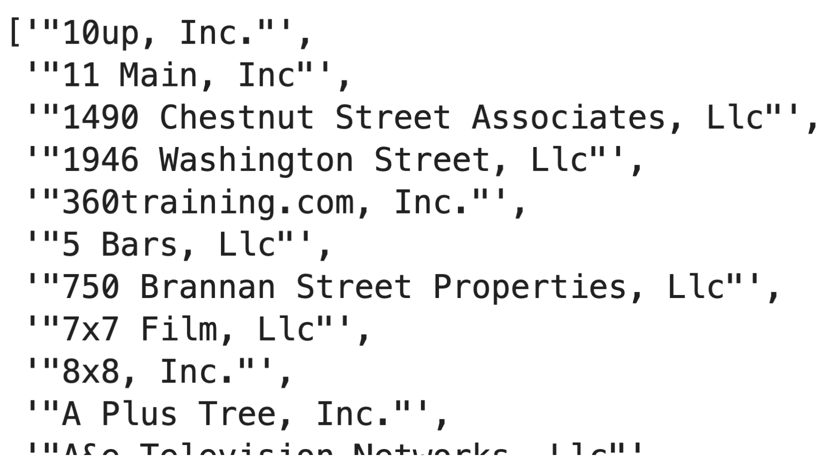
- **Class Attendance**
  - No cellphones, social media, slack, texting during the class .
  - Please only use laptops for class-related purposes.

- **Assignment**
  - No late submissions are allowed.
  - Do not share any homework and exam files - All the codes/reports including the last 6 years will be tested by Moss (Measure Of Software Similarity) and Turnitin.
  - Make sure that your code runs in Python 3 and Pyspark 3.

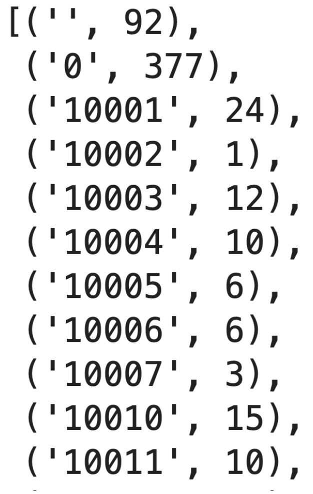
Today’s Example

From data/SF_business,

- Return unique business names without a supervisor
- Count number of business per zip code ordered by zip



```
['"10up, Inc."',
 '"11 Main, Inc"',
 '"1490 Chestnut Street Associates, Llc"',
 '"1946 Washington Street, Llc"',
 '"360training.com, Inc."',
 '"5 Bars, Llc"',
 '"750 Brannan Street Properties, Llc"',
 '"7x7 Film, Llc"',
 '"8x8, Inc."',
 '"A Plus Tree, Inc."',
 '"A&e Television Networks, Llc"',
```



```
[('', 92),
 ('0', 377),
 ('10001', 24),
 ('10002', 1),
 ('10003', 12),
 ('10004', 10),
 ('10005', 6),
 ('10006', 6),
 ('10007', 3),
 ('10010', 15),
 ('10011', 10),
```

Contents

Pair RDDs

Pair RDD Operations

- Transformation
- Action

Loading files from a directory as a pair RDD

Contents

Pair RDDs

Pair RDD Operations

- Transformation
- Action

Loading files from a directory as a pair RDD

Pair RDDs

Definition

- Key-value pairs – Commonly used for many operations including aggregations, ETL (extract, transform, and load) in Spark.
- Allow operations on each key in parallel or regroup data across the network such as reduceByKey(), join(), etc.

Key

Value

Pair RDDs

Creation

- Apply a map function with a lambda or user-defined function to have a pair of (Key, Value).
  - Key : could be a simple object (integer, string, etc.) to complex objects (tuples, etc.).
  - Value : could be a simple objects to data structures (lists, tuples, dictionaries, sets, etc.).

Example 1

Load data from SF_business/supervisor_sf.tsv and SF_business/ filtered_registered_business_sf.tsv to create

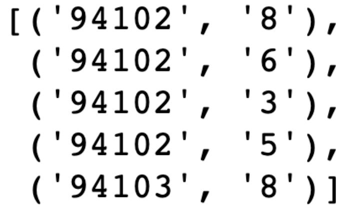
1. Pair RDD of (zip, supervisor_id) in a (string, string) format.

2. Pair RDD of (zip, [name, street, city, …]) in a (string, array) format.

Example 1

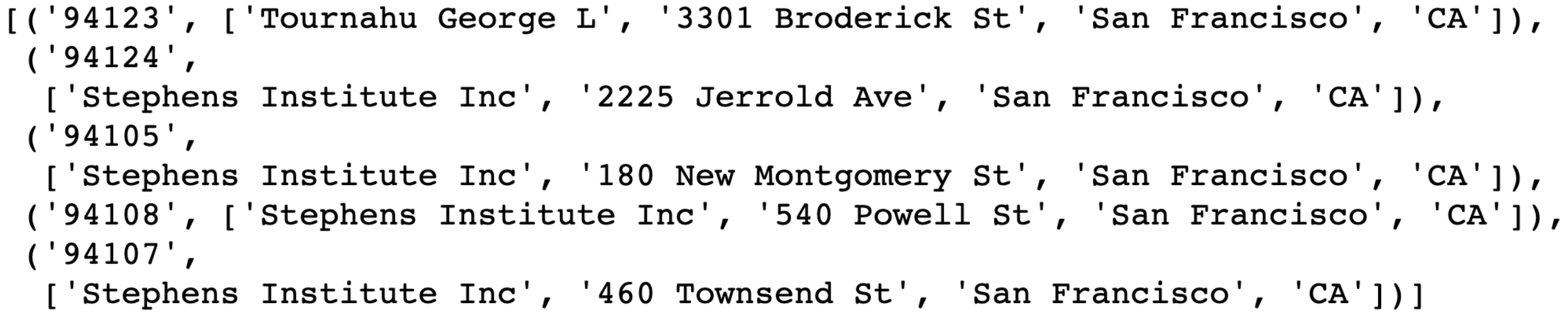
Load data from SF_business/supervisor_sf.tsv and SF_business/ filtered_registered_business_sf.tsv to create

1. Pair RDD of (zip, supervisor_id) in a (string, string) format.



```
[('94102', '8'),
 ('94102', '6'),
 ('94102', '3'),
 ('94102', '5'),
 ('94103', '8')]
```

2. Pair RDD of (zip, [name, street, city, …]) in a (string, array) format.

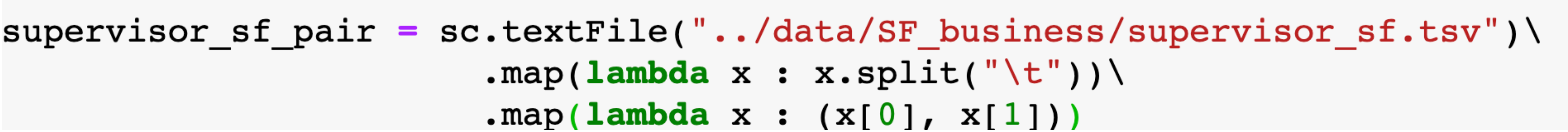


```
[('94123', ['Tournahu George L', '3301 Broderick St', 'San Francisco', 'CA']),
 ('94124',
  ['Stephens Institute Inc', '2225 Jerrold Ave', 'San Francisco', 'CA']),
 ('94105',
  ['Stephens Institute Inc', '180 New Montgomery St', 'San Francisco', 'CA']),
 ('94108', ['Stephens Institute Inc', '540 Powell St', 'San Francisco', 'CA']),
 ('94107',
  ['Stephens Institute Inc', '460 Townsend St', 'San Francisco', 'CA'])]
```

Example 1

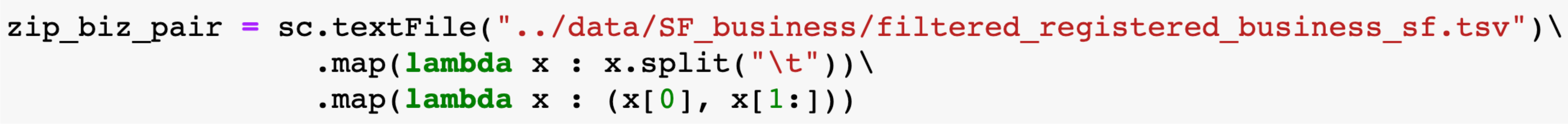
Load data from SF_business/supervisor_sf.tsv and SF_business/filtered_registered_business_sf.tsv to create

1. Pair RDD of (zip, supervisor_id) in a (string, string) format.



```
supervisor_sf_pair = sc.textFile("../data/SF_business/supervisor_sf.tsv")\
                        .map(lambda x : x.split("\t"))\
                        .map(lambda x : (x[0], x[1]))
```

2. Pair RDD of (zip, [name, street, city, …]) in a (string, array) format.



```
zip_biz_pair = sc.textFile("../data/SF_business/filtered_registered_business_sf.tsv")\
                .map(lambda x : x.split("\t"))\
                .map(lambda x : (x[0], x[1:]))
```

Contents

Pair RDDs

Pair RDD Operations

- Transformation
- Action

Loading files from a directory as a pair RDD

Pair RDDs - Operations

Most Spark operation works for pair RDDs, but there are several operators only available on RDDs of key-value pairs. The most common ones are distributed “shuffle” operations, such as grouping or aggregating the elements **by a key**.

- **Transformations**
- Actions

http://spark.apache.org/docs/latest/programming-guide.html
https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD
https://spark.apache.org/docs/0.6.2/api/core/spark/PairRDDFunctions.html

Pair RDDs - Operations

Transformation on Pair RDDs

Function name	Purpose
keys()	Return an RDD of just the keys.
values()	Return an RDD of just the values.
sortByKey()	Return an RDD sorted by the key.
groupByKey()	Group values with the same key.
mapValues(func)	Apply a function to each value of a pair RDD without changing the key.
reduceByKey(func)	Combine values with the same key.
subtractByKey(otherDataset)	Remove elements with a key present in the other RDD.
join(otherDataset)	Perform an inner join between two RDDs.
rightOuterJoin(otherDataset)	Perform a join between two RDDs where the key must be present in the first RDD.
leftOuterJoin(otherDataset)	Perform a join between two RDDs where the key must be present in the other RDD.

http://spark.apache.org/docs/latest/programming-guide.html
https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD
https://spark.apache.org/docs/0.6.2/api/core/spark/PairRDDFunctions.html

Pair RDDs - Operations

Transformation on Pair RDDs

- keys() - Return an RDD of just the keys.
- values() - Return an RDD of just the values.
- sortByKey() - Return an RDD sorted by the key (Default: ascending=True).
- groupByKey()
  - Group data using the key.
  - Return an RDD of (Key, ResultIterable) pairs.

http://spark.apache.org/docs/latest/programming-guide.html
https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD
https://spark.apache.org/docs/0.6.2/api/core/spark/PairRDDFunctions.html

Pair RDDs - Operations

Transformation on Pair RDDs

- mapValues(func)
  - Pass each value in the key-value pair RDD through a map function without changing the keys.
  - Retain the original RDD’s partitioning.

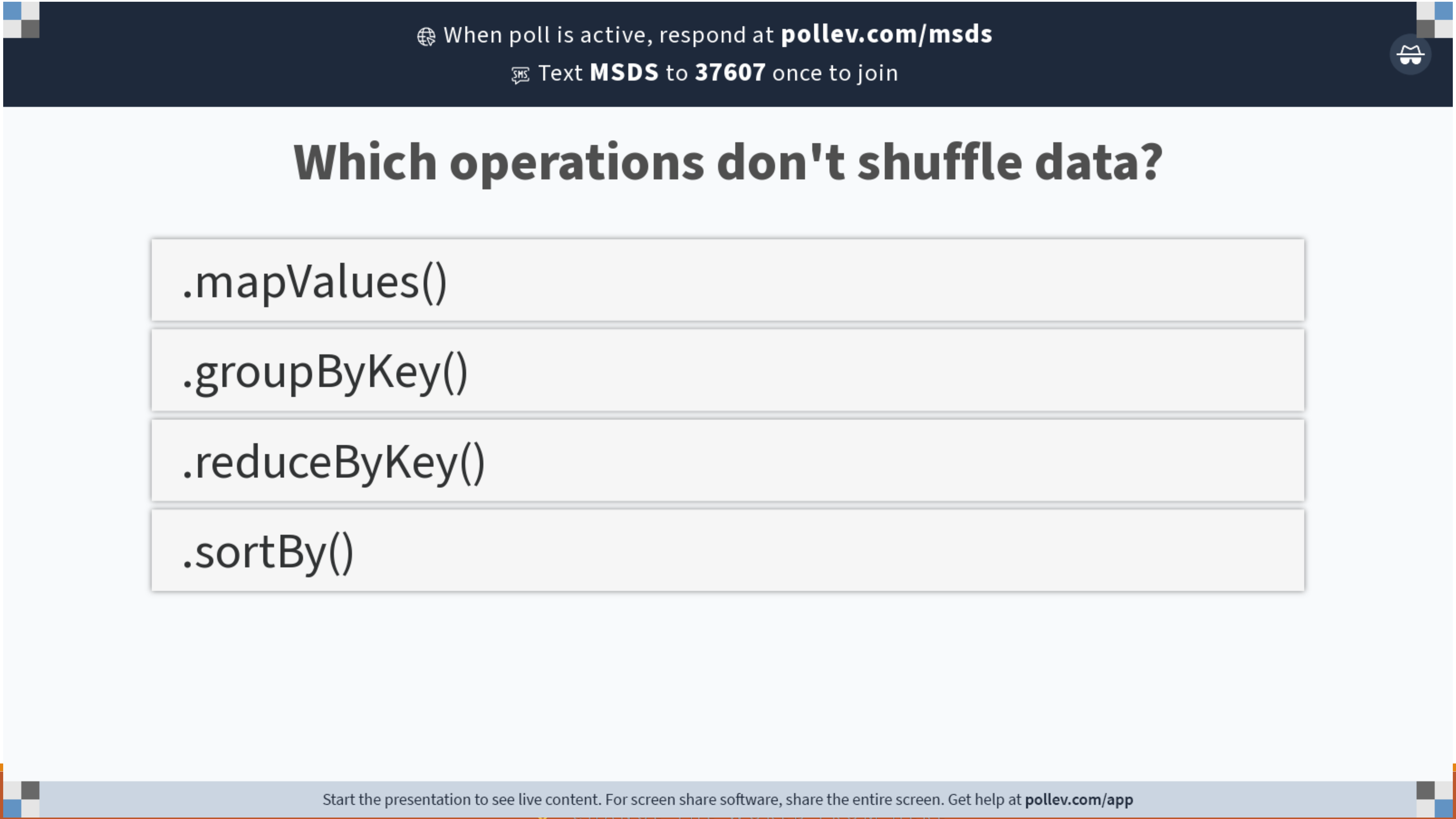
http://spark.apache.org/docs/latest/programming-guide.html
https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD
https://spark.apache.org/docs/0.6.2/api/core/spark/PairRDDFunctions.html

Pair RDDs - Operations

Transformation on Pair RDDs

- reduceByKey(func)
  - Similar to reduce().
  - Run several parallel reduce operations, one for each key in the data set.
  - When called on a dataset of (Key, Val) pairs, returns a dataset of (Key, Val) pairs where the values for each key are aggregated using the given reduce function.
  - **Transformation** (Not an action) : Returns a new RDD consisting of each key and the reduced value for that key.

http://spark.apache.org/docs/latest/programming-guide.html
https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD
https://spark.apache.org/docs/0.6.2/api/core/spark/PairRDDFunctions.html



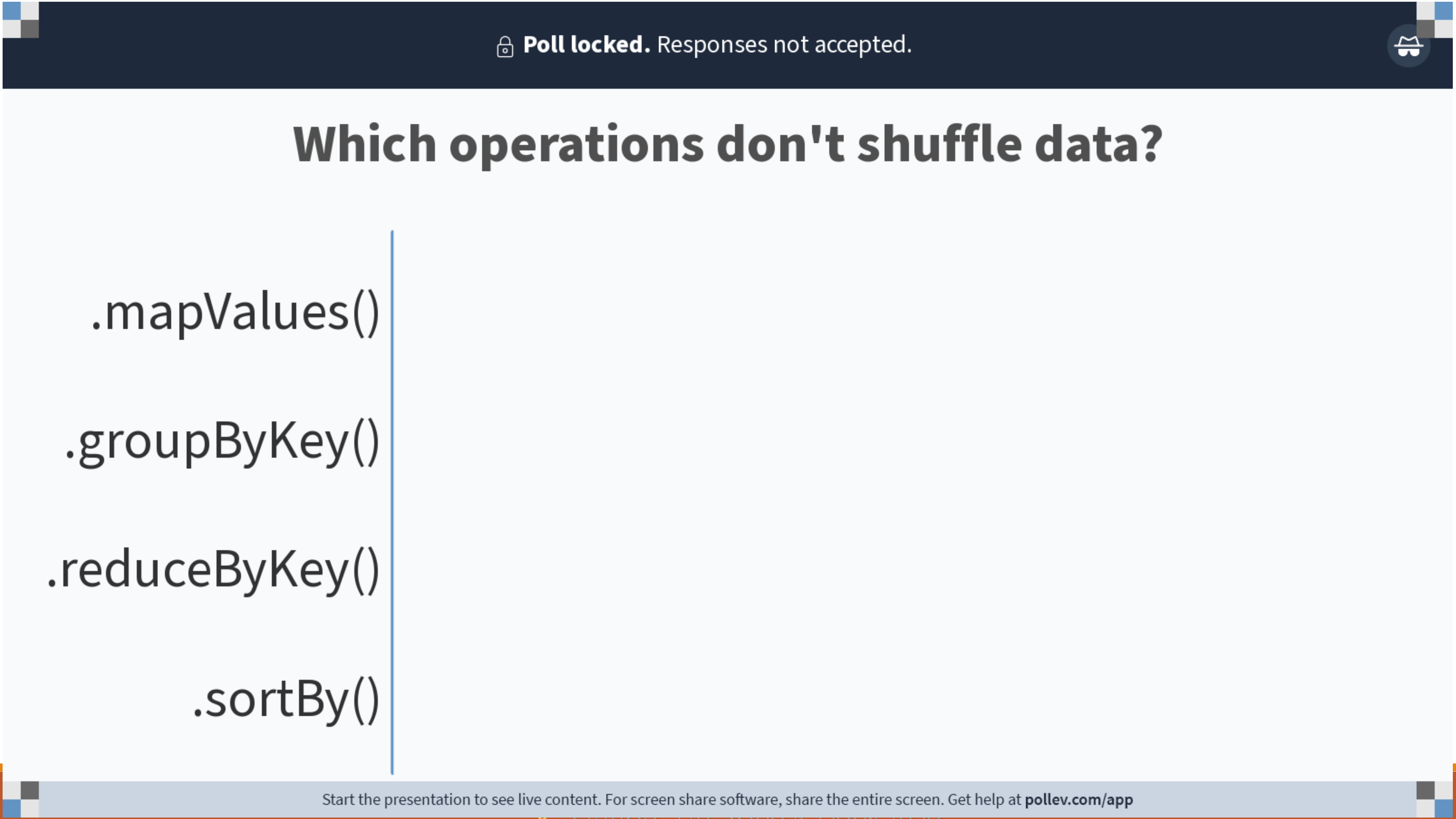
When poll is active, respond at **pollev.com/msds**

Text **MSDS** to **37607** once to join

Which operations don't shuffle data?

- .mapValues()
- .groupByKey()
- .reduceByKey()
- .sortBy()

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**



🔒 **Poll locked.** Responses not accepted.

Which operations don't shuffle data?

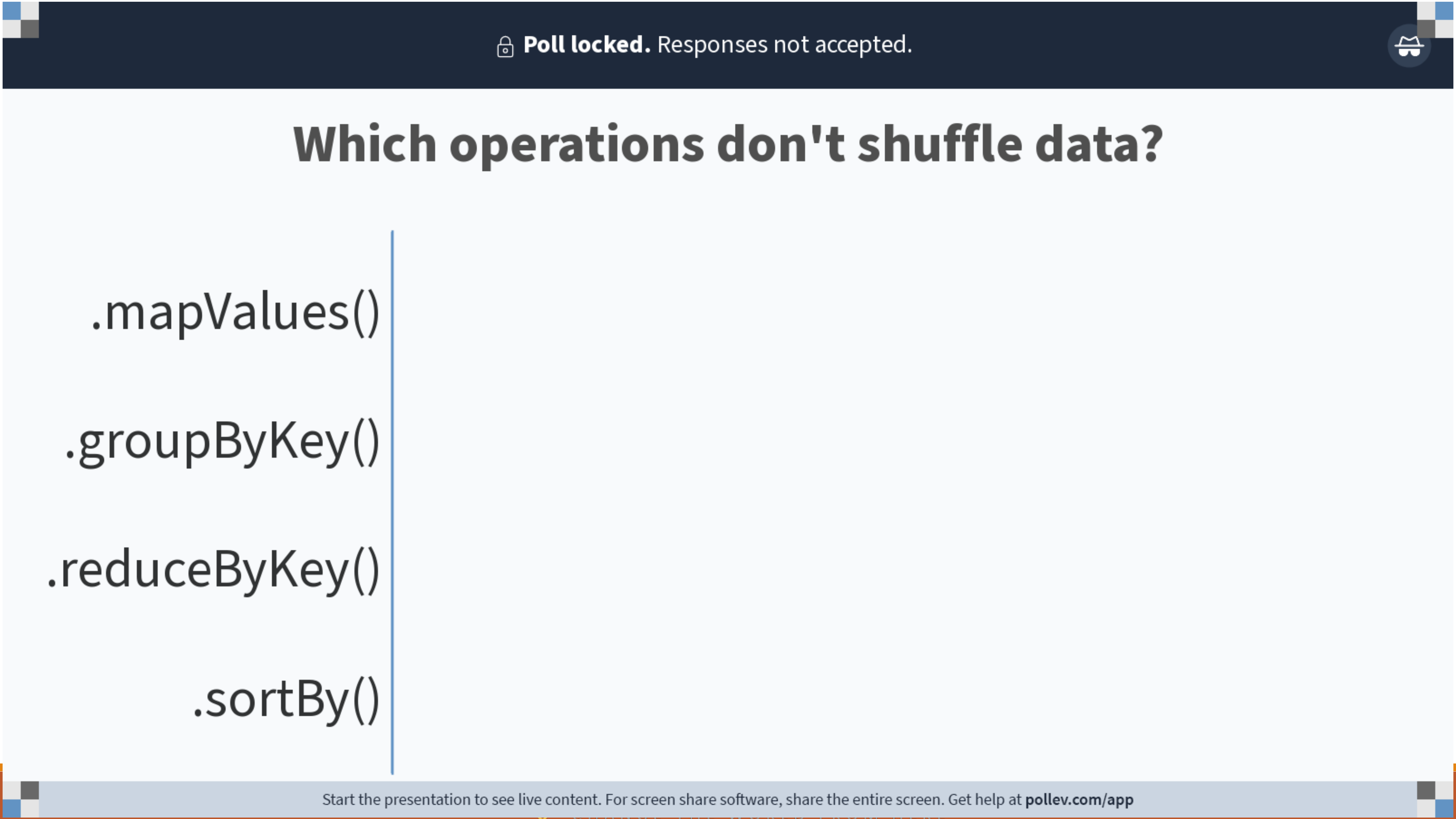
.mapValues()

.groupByKey()

.reduceByKey()

.sortBy()

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**



Poll locked. Responses not accepted.

Which operations don't shuffle data?

.mapValues()

.groupByKey()

.reduceByKey()

.sortBy()

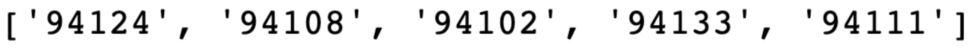
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Example 2

1. Return any five distinct zip code in zip_biz_pair.
2. Return any five distinct city names.
3. Return five distinct (zip code, city name) pairs ordered by zip code, where zip code has 5 digits.
4. Find the zip associated with the most city names.
5. Create zip_city_count_names which includes (zip, (count, the list of names))
6. Convert Q5 to (zip, city_name).

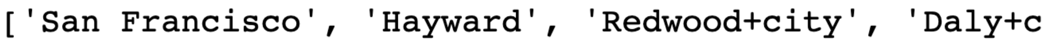
Example 2

1. Return any five distinct zip code in zip_biz_pair.



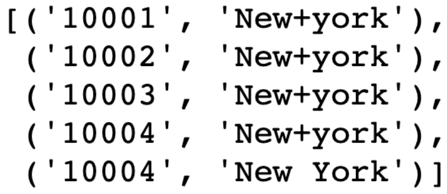
```
['94124', '94108', '94102', '94133', '94111']
```

2. Return any five distinct city names.



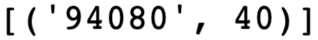
```
['San Francisco', 'Hayward', 'Redwood+city', 'Daly+c
```

3. Return five distinct (zip code, city name) pairs ordered by zip code, where zip code has 5 digits.



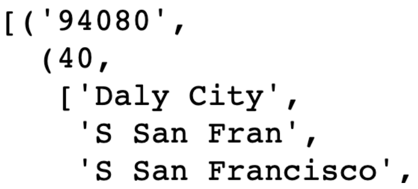
```
[('10001', 'New+york'),
 ('10002', 'New+york'),
 ('10003', 'New+york'),
 ('10004', 'New+york'),
 ('10004', 'New York')]
```

4. Find the zip associated with the most city names.



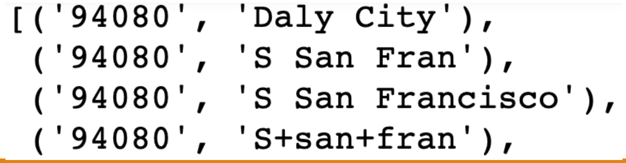
```
[('94080', 40)]
```

5. Create zip_city_count_names which includes (zip, (count, the list of names))



```
[('94080',
  (40,
   ['Daly City',
    'S San Fran',
    'S San Francisco',
```

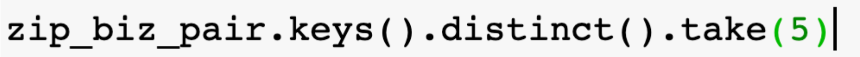
6. Convert Q5 to (zip, city_name).



```
[('94080', 'Daly City'),
 ('94080', 'S San Fran'),
 ('94080', 'S San Francisco'),
 ('94080', 'S+san+fran'),
```

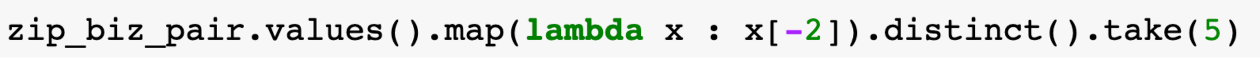
Example 2

1. Return any five distinct zip code in zip_biz_pair.



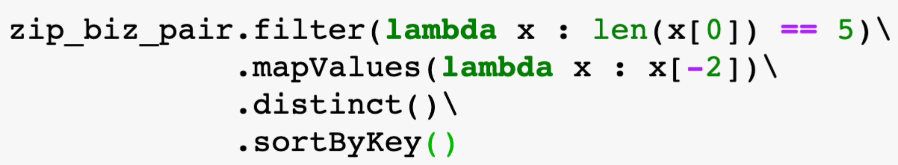
```
zip_biz_pair.keys().distinct().take(5)
```

2. Return any five distinct city names.



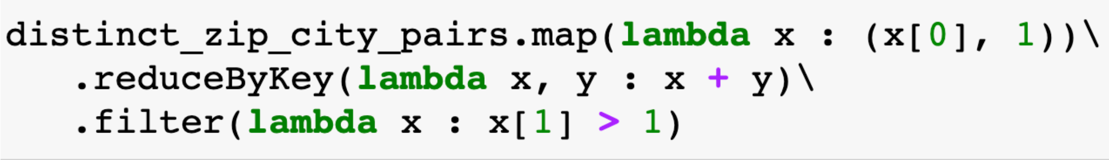
```
zip_biz_pair.values().map(lambda x : x[-2]).distinct().take(5)
```

3. Return five distinct (zip code, city name) pairs ordered by zip code, where zip code has 5 digits.



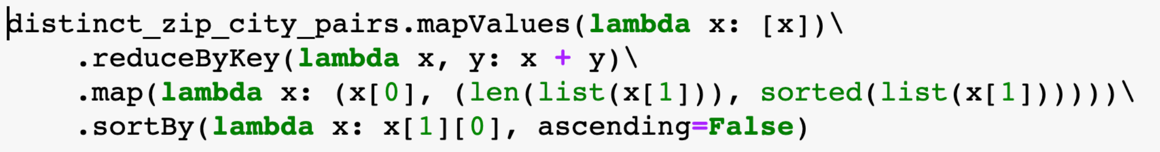
```
zip_biz_pair.filter(lambda x : len(x[0]) == 5)\
                .mapValues(lambda x : x[-2])\
                .distinct()\
                .sortByKey()
```

4. Find the zip associated with the most city names.



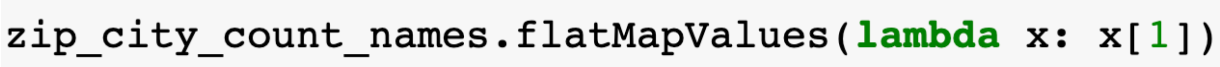
```
distinct_zip_city_pairs.map(lambda x : (x[0], 1))\
    .reduceByKey(lambda x, y : x + y)\
    .filter(lambda x : x[1] > 1)
```

5. Create zip_city_count_names which includes (zip, (count, the list of names))



```
distinct_zip_city_pairs.mapValues(lambda x: [x])\
    .reduceByKey(lambda x, y: x + y)\
    .map(lambda x: (x[0], (len(list(x[1])), sorted(list(x[1])))))\
    .sortBy(lambda x: x[1][0], ascending=False)
```

6. Convert Q5 to (zip, city_name).



```
zip_city_count_names.flatMapValues(lambda x: x[1])
```

Example 2

4. Find the zip associated with the most city names.

v1.

```
distinct_zip_city_pairs.map(lambda x : (x[0], 1))\
                      .groupByKey()\
                      .mapValues(lambda x : sum(x))\
                      .sortBy(lambda x : x[1], ascending = False)
                      .take(1)
```

v2.

```
distinct_zip_city_pairs.map(lambda x : (x[0], 1))\
                       .reduceByKey(lambda x, y : x + y)\
                       .sortBy(lambda x : x[1], ascending = False)\
                       .take(1)
```

When poll is active, respond at **pollev.com/msds**

Text **MSDS** to **37607** once to join

Which of the following is more efficient?

```
rdd.groupByKey()
.mapValues(func)
```

```
rdd.reduceByKey(func)
```

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Which of the following is more efficient?

```
rdd.groupByKey()
.mapValues(func)
```

```
rdd.reduceByKey(func)
```

When poll is active, respond at **pollev.com/msds**

Text **MSDS** to **37607** once to join

Which of the following is more efficient?

```
rdd.groupByKey()
.mapValues(func)
```

```
rdd.reduceByKey(func)
```

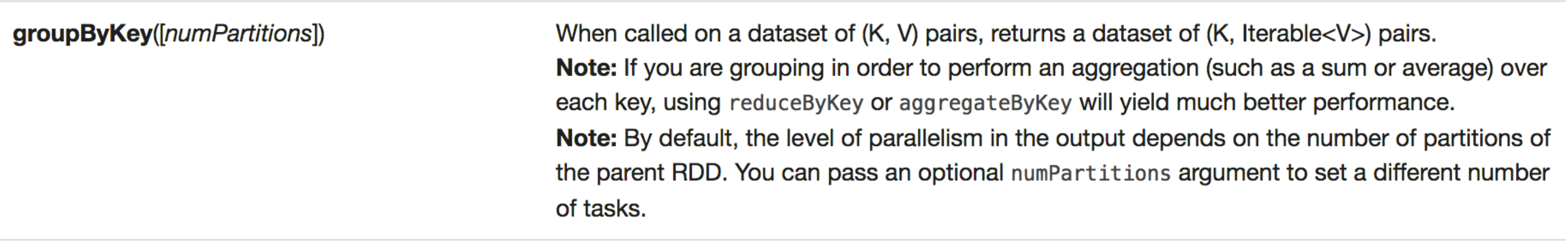
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Example 2.5 - v1 or v2?

To group values for the purpose of aggregation (such as sum or count for each key), using reduceByKey(), foldByKey() will provide better performance.

- They combines the aggregation function before the shuffle.
  - Pairs on the same node/partition with the same key are combined before the data is shuffled. Then the function is called again to reduce all the values from each partition to produce one final result.

➔ Result in a reduced amount of data shuffled.



groupByKey([*numPartitions*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. **Note:** If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using `reduceByKey` or `aggregateByKey` will yield much better performance. **Note:** By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional `numPartitions` argument to set a different number of tasks.

https://databricks.gitbooks.io/databricks-spark-knowledge-base/content/best_practices/prefer_reducebykey_over_groupbykey.html
https://spark.apache.org/docs/latest/rdd-programming-guide.html

Pair RDDs - Operations

Transformation on Two Pair RDDs

- subtractByKey(otherDataset)
  - Return an RDD with the pairs whose keys are not in otherDataset.
- join(otherDataset)
  - Perform an inner join between two RDDs.
- rightOuterJoin(otherDataset)
  - Perform a join between two RDDs where the key must be present in the other RDD.
- leftOuterJoin(otherDataset)
  - Perform a join between two RDDs where the key must be present in the first RDD.

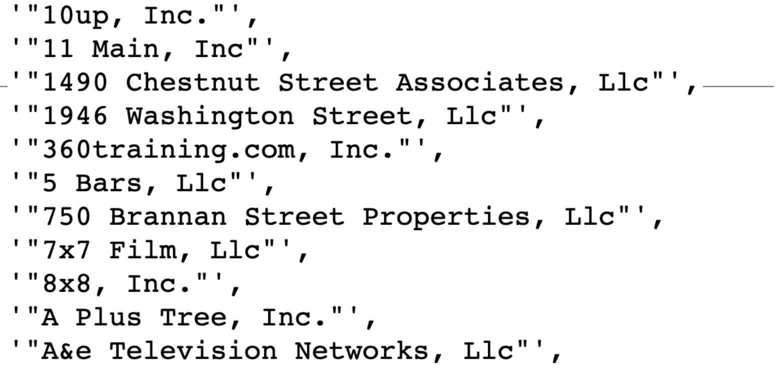
http://spark.apache.org/docs/latest/programming-guide.html
https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD
https://spark.apache.org/docs/0.6.2/api/core/spark/PairRDDFunctions.html

Example 3

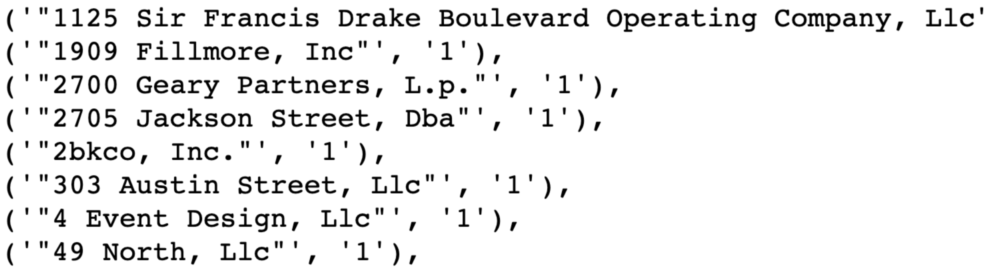
1. Using “filtered_registered_business_sf.tsv” and “supervisor_sf.tsv”, list unique business names without a supervisor ordered by name.
2. Create pairs of business name and supervisor id ordered by supervisor id (first) and business name (second).
3. Create pairs of business and supervisor id for all the business ordered by supervisor id (first) and business name (second). (although it may not have a district supervisor).

Example 3

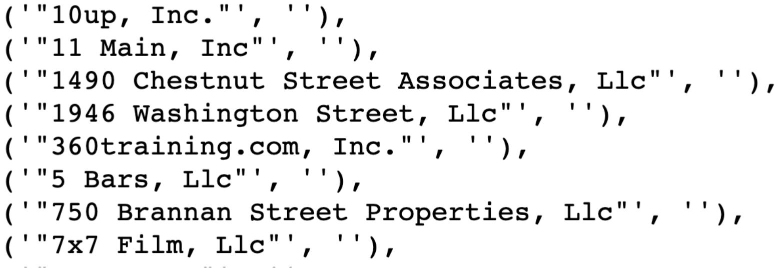
1. Using “filtered_registered_business_sf.tsv” and “supervisor_sf.tsv”, list unique business names without a supervisor ordered by name.
2. Create pairs of business name and supervisor id ordered by supervisor id (first) and business name (second).
3. Create pairs of business and supervisor id for all the business ordered by supervisor id (first) and business name (second). (although it may not have a district supervisor).



```
'"10up, Inc."',
'"11 Main, Inc"',
'"1490 Chestnut Street Associates, Llc"',
'"1946 Washington Street, Llc"',
'"360training.com, Inc."',
'"5 Bars, Llc"',
'"750 Brannan Street Properties, Llc"',
'"7x7 Film, Llc"',
'"8x8, Inc."',
'"A Plus Tree, Inc."',
'"A&e Television Networks, Llc"',
```



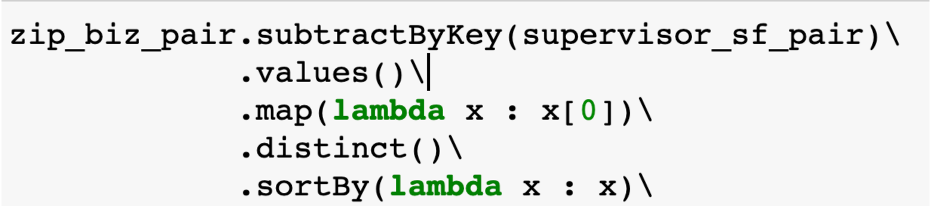
```
('"1125 Sir Francis Drake Boulevard Operating Company, Llc"
('"1909 Fillmore, Inc"', '1'),
('"2700 Geary Partners, L.p."', '1'),
('"2705 Jackson Street, Dba"', '1'),
('"2bkco, Inc."', '1'),
('"303 Austin Street, Llc"', '1'),
('"4 Event Design, Llc"', '1'),
('"49 North, Llc"', '1'),
```



```
('"10up, Inc."', ''),
('"11 Main, Inc"', ''),
('"1490 Chestnut Street Associates, Llc"', ''),
('"1946 Washington Street, Llc"', ''),
('"360training.com, Inc."', ''),
('"5 Bars, Llc"', ''),
('"750 Brannan Street Properties, Llc"', ''),
('"7x7 Film, Llc"', ''),
```

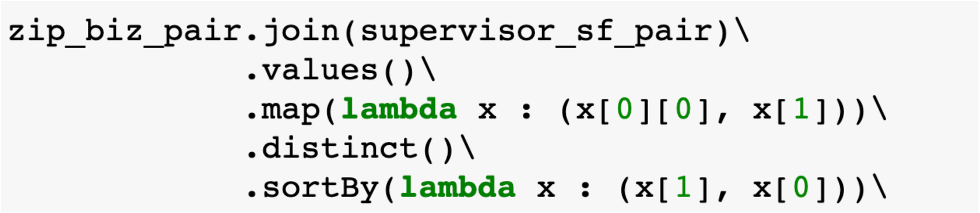
Example 3

1. Using “filtered_registered_business_sf.tsv” and “supervisor_sf.tsv”, list unique business names without a supervisor ordered by name.



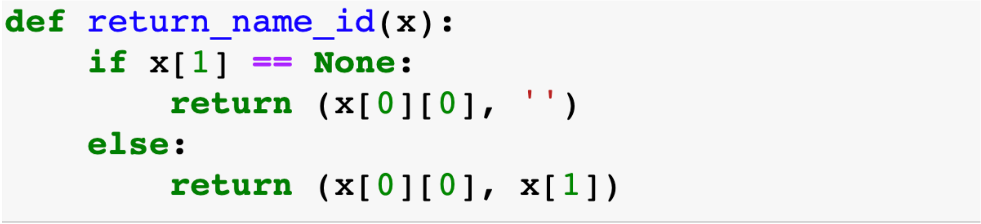
```
zip_biz_pair.subtractByKey(supervisor_sf_pair)\
              .values()\
              .map(lambda x : x[0])\
              .distinct()\
              .sortBy(lambda x : x)\
```

2. Create pairs of business name and supervisor id ordered by supervisor id (first) and business name (second).

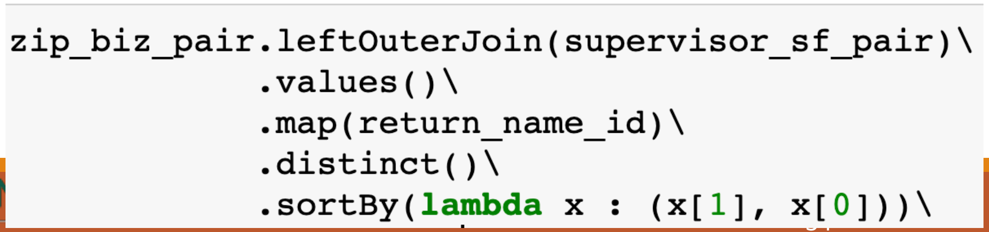


```
zip_biz_pair.join(supervisor_sf_pair)\
              .values()\
              .map(lambda x : (x[0][0], x[1]))\
              .distinct()\
              .sortBy(lambda x : (x[1], x[0]))\
```

3. Create pairs of business and supervisor id for all the business ordered by supervisor id (first) and business name (second). (although it may not have a district supervisor).



```
def return_name_id(x):
    if x[1] == None:
        return (x[0][0], '')
    else:
        return (x[0][0], x[1])
```



```
zip_biz_pair.leftOuterJoin(supervisor_sf_pair)\
              .values()\
              .map(return_name_id)\
              .distinct()\
              .sortBy(lambda x : (x[1], x[0]))\
```

Contents

Pair RDDs

Pair RDD Operations

- Transformation
- **Action**

Loading files from a directory as a pair RDD

Pair RDDs - Operations

Most Spark operations work for pair RDDs, but there are several operators only available on RDDs of key-value pairs. The most common ones are distributed “shuffle” operations, such as grouping or aggregating the elements by a key.

- Transformations
- **Actions**

http://spark.apache.org/docs/latest/programming-guide.html
https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD
https://spark.apache.org/docs/0.6.2/api/core/spark/PairRDDFunctions.html

Pair RDDs - Operations

Actions on Pair RDDs

countByKey()	Only available on pair RDDs. Returns (Key, Int) pairs with the count of each key.
lookup(key)	Return all values associated with the provided key.

https://spark.apache.org/docs/latest/rdd-programming-guide.html#working-with-key-value-pairs
https://spark.apache.org/docs/0.6.2/api/core/spark/PairRDDFunctions.html

Example 4

Using “filtered_registered_business_sf.tsv”,

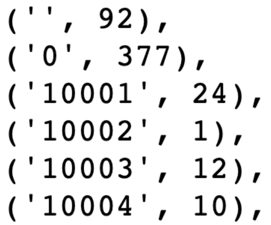
1. Count number of business per zip code.

2. Count the number of businesses which do not have a zip (empty zip) as a key.

Example 4

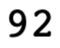
Using “filtered_registered_business_sf.tsv”,

1. Count number of business per zip code ordered by zip.



```
('', 92),
('0', 377),
('10001', 24),
('10002', 1),
('10003', 12),
('10004', 10),
```

2. Count the number of businesses which do not have a zip (empty zip) as a key.

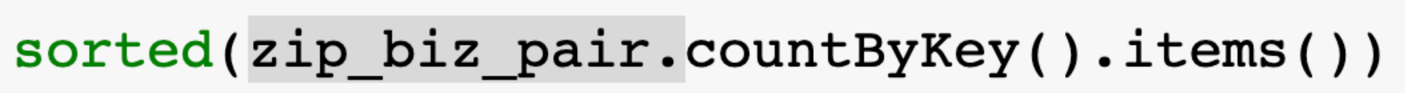


```
92
```

Example 4

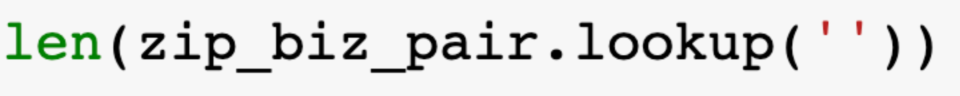
Using “filtered_registered_business_sf.tsv”,

1. Count number of business per zip code ordered by zip.



```
sorted(zip_biz_pair.countByKey().items())
```

2. Count the number of businesses which do not have a zip (empty zip) as a key.



```
len(zip_biz_pair.lookup(''))
```

Contents

Pair RDDs

Pair RDD Operations

- Transformation
- Action

Loading files from a directory as a pair RDD

Loading and Saving Data

File Formats

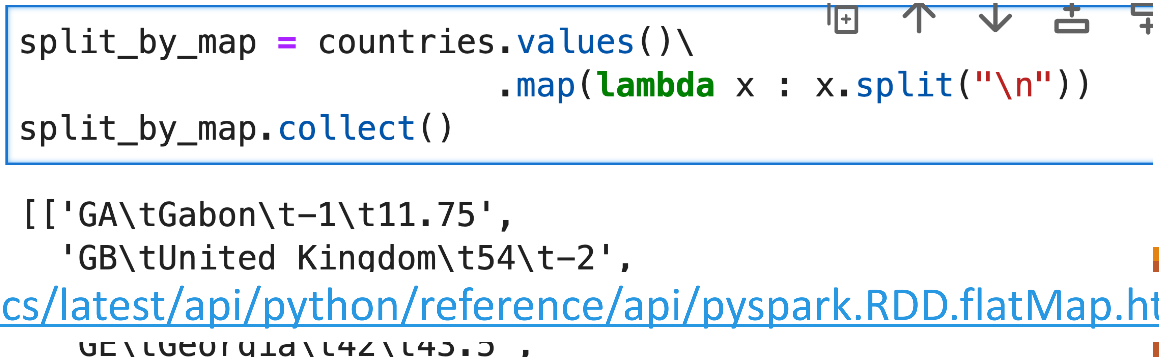
Text Format

- Load
  - One line is one element in a single file.
    - Use **textFile(file_name or folder_name)** method.
  - Can load multiple text files as **pair RDDs**, with the key being the file name and the value being the file content.
    - Use **wholeTextFiles(dir_name).**
    - Useful when each file represent a certain time period's data and need to use it for statistics. Ex. sales stat, sensor readings.
- Save
  - Use **saveAsTextFile(new_subdir_name)** method.
  - Return multiple output files underneath the new_subdir_name, as Spark writes the output from multiple partitions/nodes.

Loading and Saving Data

WholeTextFile(dir_name)

- Creates an RDD of (key, value) format where the key is a file name and the value is the content of the file.
  - Ex. `[(“file_name1”, “file_contents”), (“file_name2”, “file_contents), …]`
  - To retrieve all the `file_contents` only
    - `file_contents` are not parsed by new lines.
    - Rather than using `.map(lambda x : x.split(“\n”))`, `.flatMap(lambda x : x.split(“\n”))` would split the contents by new lines and flatten the structure.



```
split_by_map = countries.values()\
                        .map(lambda x : x.split("\n"))
split_by_map.collect()
```

```
[['GA\tGabon\t-1\t11.75',
  'GB\tUnited Kingdom\t54\t-2',
  'GE\tGeorgia\t42\t43.5',
```

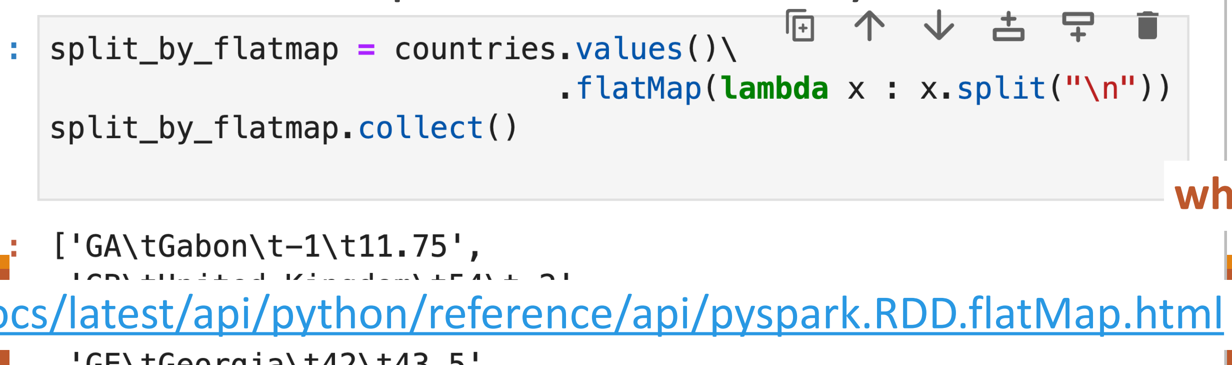
wholeTextFile.ipynb

https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.flatMap.html

Loading and Saving Data

WholeTextFile(dir_name)

- Creates an RDD of (key, value) format where the key is a file name and the value is the content of the file.
  - Ex. `[(“file_name1”, “file_contents”), (“file_name2”, “file_contents), …]`
  - To retrieve all the `file_contents` only
    - `file_contents` are not parsed by new lines.
    - Rather than using `.map(lambda x : x.split(“\n”))`, `.flatMap(lambda x : x.split(“\n”))` would split the contents by new lines and flatten the structure.



```
: split_by_flatmap = countries.values()\
                              .flatMap(lambda x : x.split("\n"))
  split_by_flatmap.collect()
```

```
: ['GA\tGabon\t-1\t11.75',
   'GB\tUnited Kingdom\t54\t-2'
   'GE\tGeorgia\t42\t43.5'
```

wholeTextFile.ipynb

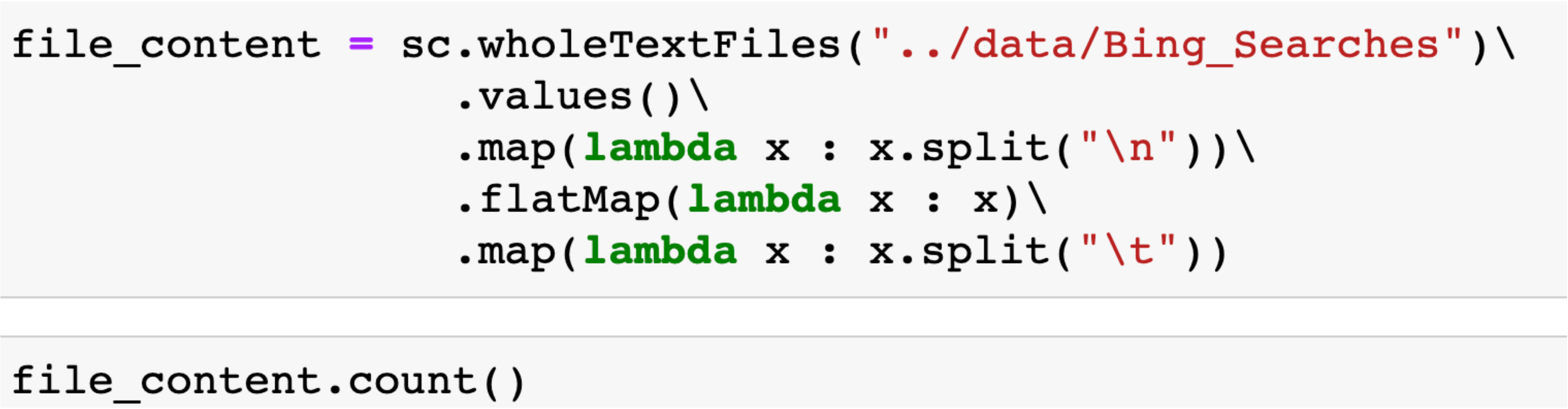
https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.RDD.flatMap.html

Example 5

Load all the files in data/Bing_Searches and return the number of lines.

Example 5

Load all the files in data/Bing_Searches and return the number of lines.



```
file_content = sc.wholeTextFiles("../data/Bing_Searches")\
                  .values()\
                  .map(lambda x : x.split("\n"))\
                  .flatMap(lambda x : x)\
                  .map(lambda x : x.split("\t"))
```

```
file_content.count()
```

Contents

Pair RDDs

Pair RDD Operations

- Transformation
- Action

Loading files from a directory as a pair RDD

Reference

Spark Online Documentation, http://spark.apache.org/docs/latest/

Zecevic, Petar, et al. Spark in Action, Manning, 2016.