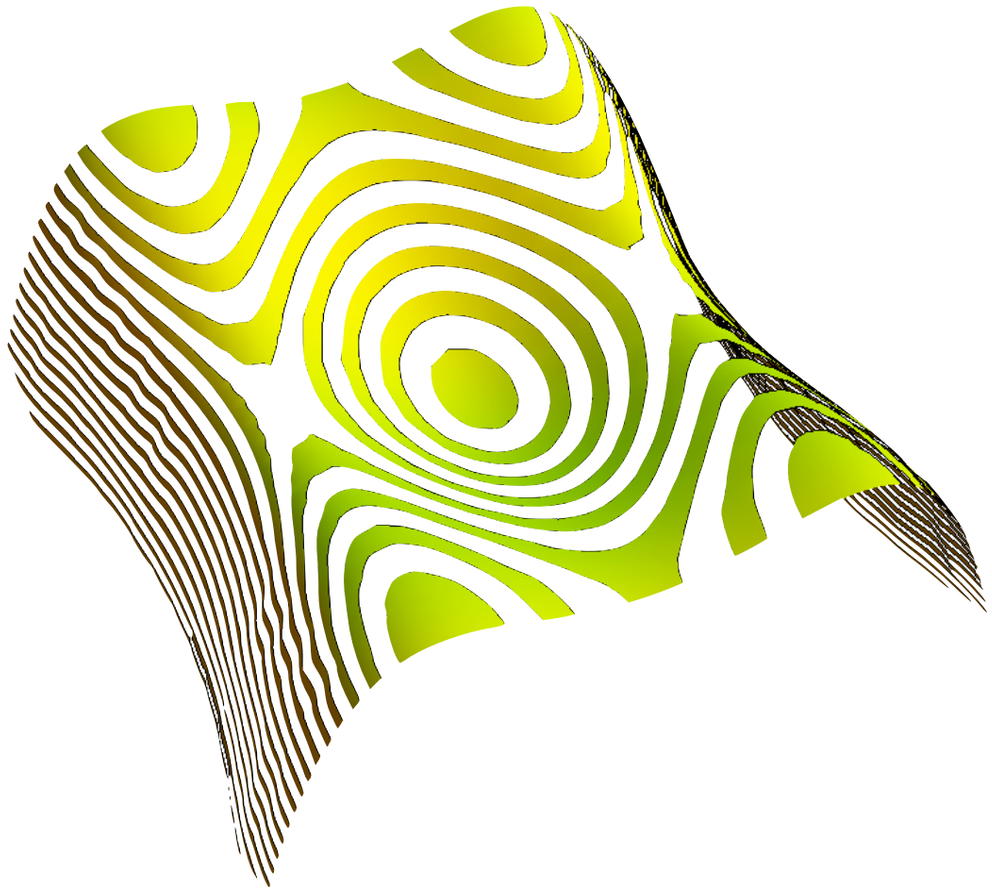
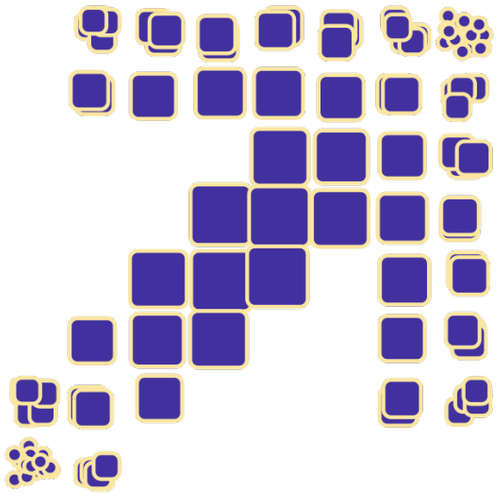




pysparkling v0.3

A pure Python implementation of Spark's RDD interface.

Sven Kreiss
New York, August 15-16, 2015

I am a Data Scientist at Wildcard. We launched last Tuesday and are currently featured in the App Store as “Best New App”.

We are looking to grow our data engineering team.

Weaknesses

Big Data processing
(use Spark)

distributed sort
(use Spark)

Strengths

small data

Python microservice
backend
(latency, dependencies)

local development
environment

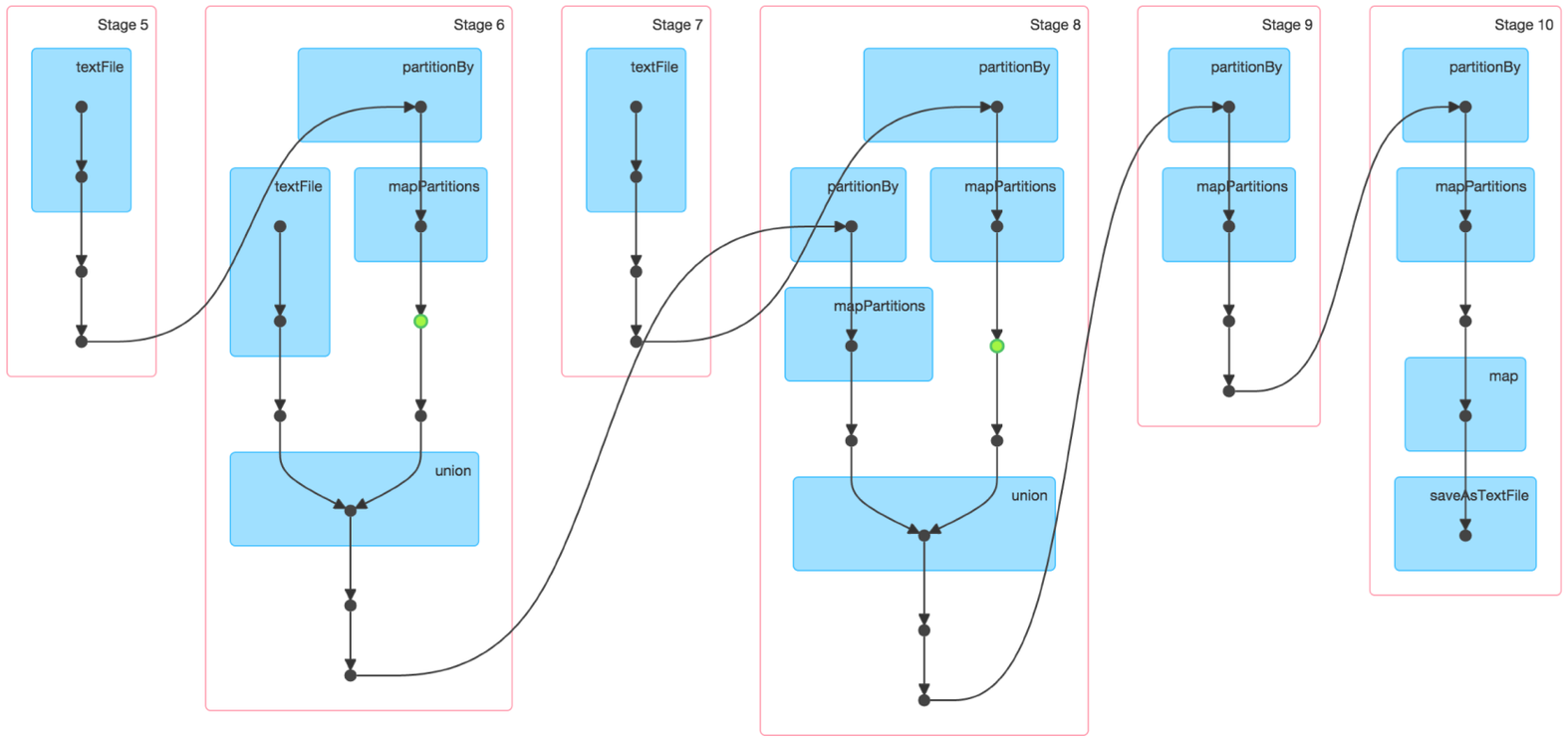
backend for spot checking
data tool

Example Data Pipeline

Some details make this pipeline more complicated than simple maps: **joins** with labeled truth, **random splits** for train-test-split, **failure resolution** for scraping, **caching**.

Example: Join Two Datasets by URL

Complication: the first dataset contains records with redirected and original URLs and the second dataset is keyed only by one URL, but it can be either.



pysparkling.fileio

Read lines from a text file:

```
> c = pysparkling.Context()
>
> rdd = c.textFile('my_textfile.txt')
> rdd = c.textFile('my_textfile.txt.gz')
> rdd = c.textFile('my_textfile.txt.bz2')
> rdd = c.textFile('http://www.svenkreiss.com/my_textfile.txt')
> rdd = c.textFile('s3n://this_bucket_does_not_exist/my_textfile.txt')
> rdd = c.textFile('hdfs://localhost/user/hadoop/my_textfile.txt.gz')
```

→ lines from a text file are read seamlessly from different locations and with different compressions. Multiple files can be specified in a comma separated list. The wildcard characters **?** and ***** are resolved.

You can use the lower level functions using the **pysparkling.fileio.File** and **pysparkling.fileio.TextFile** classes that implement the methods **load()**, **dump()** and **exists()**.

Basic Operations and Partitions

As in Spark, you have to specify the number of partitions of the data:

```
> c = pysparkling.Context()
> rdd = c.parallelize(range(100), 20)
```

creates 20 partitions of the numbers 0 … 99. Now, add 10 to every number.

```
> rdd = rdd.map(lambda n: n + 10)
```

As in Spark, all **operations are lazy** and so far, none of the maps were executed. Cache this RDD at this step once it gets evaluated.

```
> rdd = rdd.cache()
```

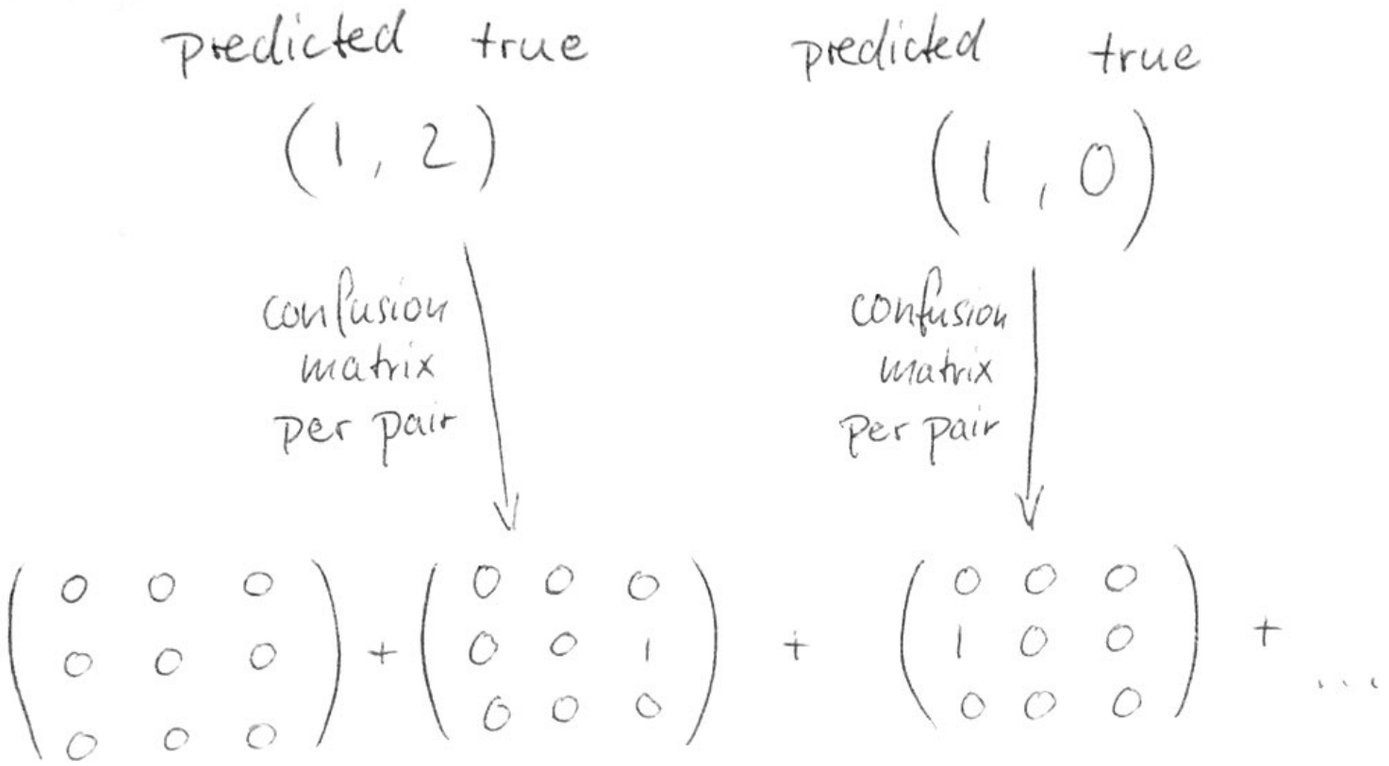
Now get the first element:

```
> f = rdd.first()
```

This triggers the computation of the first partition (and the first partition only), caches it and returns the first element from it.

Almost a Real World Example: Distributed Computation of a Confusion Matrix

Input: A map operation applied a classifier to a large number of samples. At this stage, we have pairs of predicted and true class labels for every sample.



		Predicted		
		Cat	Dog	Rabbit
Actual class	Cat	5	3	0
	Dog	2	3	1
	Rabbit	0	2	11

https://en.wikipedia.org/wiki/Confusion_matrix

Precision, Recall, Support and F-scores are simple sums and ratios of elements in the confusion table.

Almost a Real World Example:
Distributed Computation of a Confusion Matrix

sequence operation **seqOp**: pair to confusion matrix
combination operation **combOp**: sum the confusion matrices

In [11]:
```
import numpy as np

predicted_true = Context().parallelize([
    (0, 0), (1, 0), (2, 2), (1, 1), (1, 2), (2, 2),
    (0, 2), (1, 0), (2, 1), (1, 1), (0, 0), (0, 0),
])

def cm_per_pair(p_t):
    cm = np.zeros((3, 3))
    cm[p_t[0], p_t[1]] = 1
    return cm

predicted_true.aggregate(
    zeroValue=np.zeros((3, 3)),
    seqOp=lambda prev, p_t: prev + cm_per_pair(p_t),
    combOp=np.add,
)
```

Out[11]:
```
array([[ 3.,  0.,  1.],
       [ 2.,  2.,  1.],
       [ 0.,  1.,  2.]])
```

Parallel Processing (Experimental)

Initial support for **any pool instance** with a *map(iterable, func)* method.

```
> c = pysparkling.Context(
        pool=multiprocessing.Pool(7),
        serializer=cloudpickle.dumps,
        deserializer=pickle.loads,
    )
```

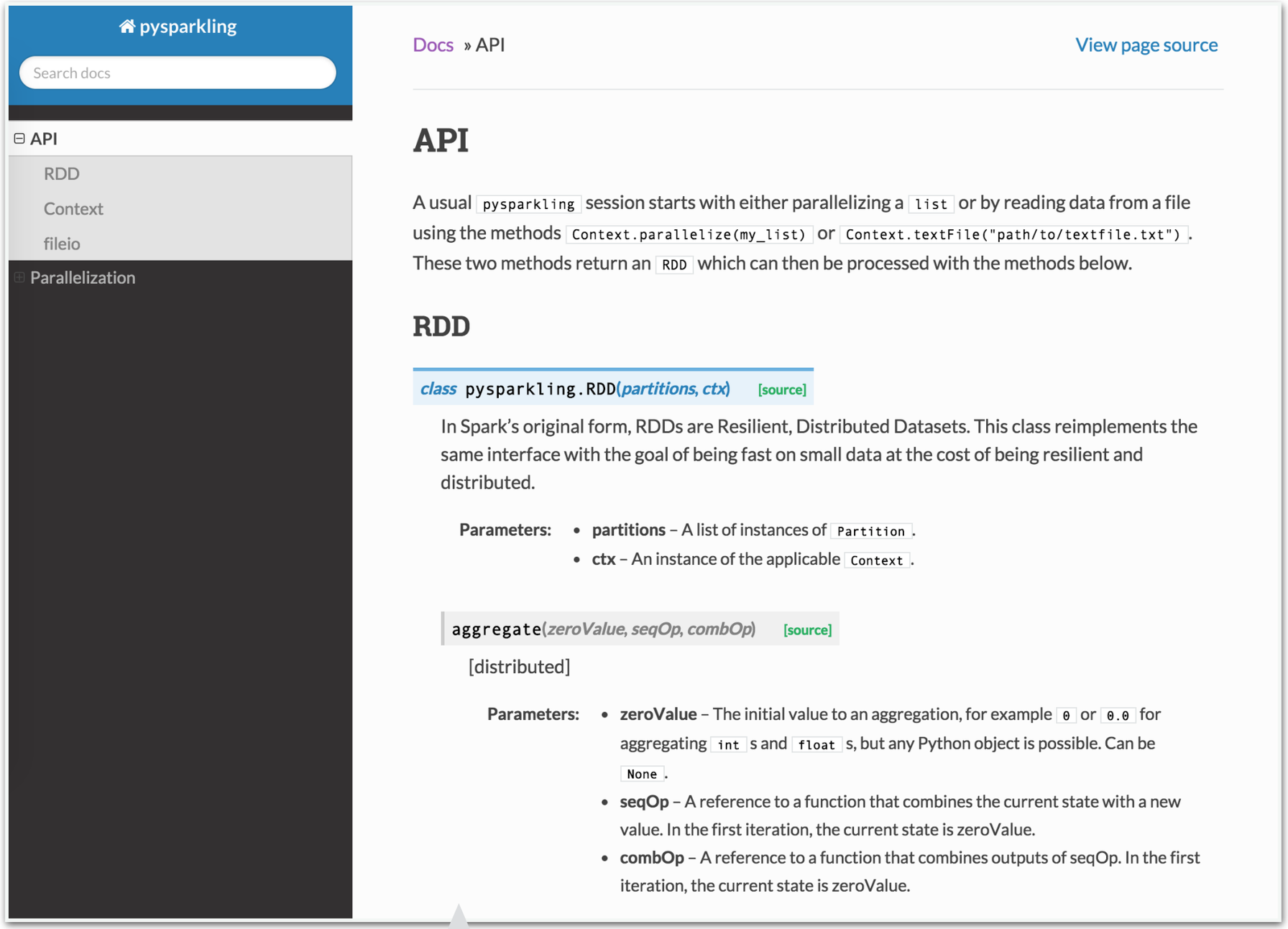
Maps are chained: applying *rdd.map()* operations consecutively results in a single multiprocessing map run.

Intermediate caches are preserved: intermediate caches in chained map operations are available for further calculations.

Other possible pool objects: futures.**ThreadPoolExecutor**, futures.**ProcessPoolExecutor**, **IPython.parallel views**.

The underlying parallelization frameworks only parallelize map operations. Any operation based on shuffles, sorts, groups, … is still run locally. Those functions are marked in the API documentation.

Documentation: API



http://pysparkling.trivial.io/v0.3/

Contains embedded example code and example output for almost every function. Those are automatically run as part of the test suite on every commit and are guaranteed to work.

Documentation: Demos

In [1]:
```
from pysparkling import Context
```

Word Count

In [2]:
```
counts = Context().textFile(
    '../README.rst'
).map(
    lambda line: ''.join(ch if ch.isalnum() else ' ' for ch in line)
).flatMap(
    lambda line: line.split(' ')
).map(
    lambda word: (word, 1)
).reduceByKey(
    lambda a, b: a + b
)
print(counts.collect())
```

```
[(u'', 1302), (u'1', 1), (u'100k', 1), (u'A', 1), (u'ACCESS', 2), (u'API', 3), (u'AWS', 3),
(u'Broadcast', 2), (u'BytesIO', 1), (u'Context', 4), (u'Count', 1), (u'Examples', 1), (u'Fa
lse', 2), (u'Features', 1), (u'File', 1), (u'HISTORY', 2), (u'Hadoop', 1), (u'Handles', 1),
(u'ID', 1), (u'If', 1), (u'Infers', 1), (u'Install', 1), (u'It', 1), (u'JVM', 2), (u'KEY',
2), (u'None', 7), (u'Now', 1), (u'Parallelization', 1), (u'Pool', 3), (u'PySpark', 1), (u'P
ython', 1), (u'RDD', 20), (u'RDDs', 1), (u'README', 1), (u'Resolves', 1), (u'S3', 5), (u'SE
CRET', 1), (u'Spark', 2), (u'SparkContext', 1), (u'Specify', 1), (u'StatCounter', 1), (u'Su
pports', 1), (u'The', 5), (u'They', 1), (u'This', 3), (u'ThreadPoolExecutor', 1), (u'To',
1), (u'URI', 1), (u'Use', 2), (u'Word', 1), (u'You', 1), (u'a', 36), (u'absolut', 1), (u'ab
solute', 1), (u'access', 1), (u'accessed', 1), (u'advanced', 1), (u'aggregate', 5), (u'aggr
egateByKey', 1), (u'all', 1), (u'an', 6), (u'and', 26), (u'any', 1), (u'apply', 5), (u'ar
e', 6), (u'as', 3), (u'auth', 1), (u'b', 2), (u'badge', 1), (u'bash', 1), (u'be', 3), (u'bl
ob', 2), (u'block', 2), (u'boto', 1), (u'broadcast', 1), (u'bucket', 2), (u'buckets', 2),
```

https://github.com/svenkreiss/pysparkling/blob/master/docs/demo.ipynb

Install:

```
$ pip install pysparkling[s3,http,hdfs]
```

Documentation: pysparkling.trivial.io

Github: https://github.com/svenkreiss/pysparkling
contribute questions, issues, pull requests, documentation, examples

Slides: trivial.io

@svenkreiss
me@svenkreiss.com