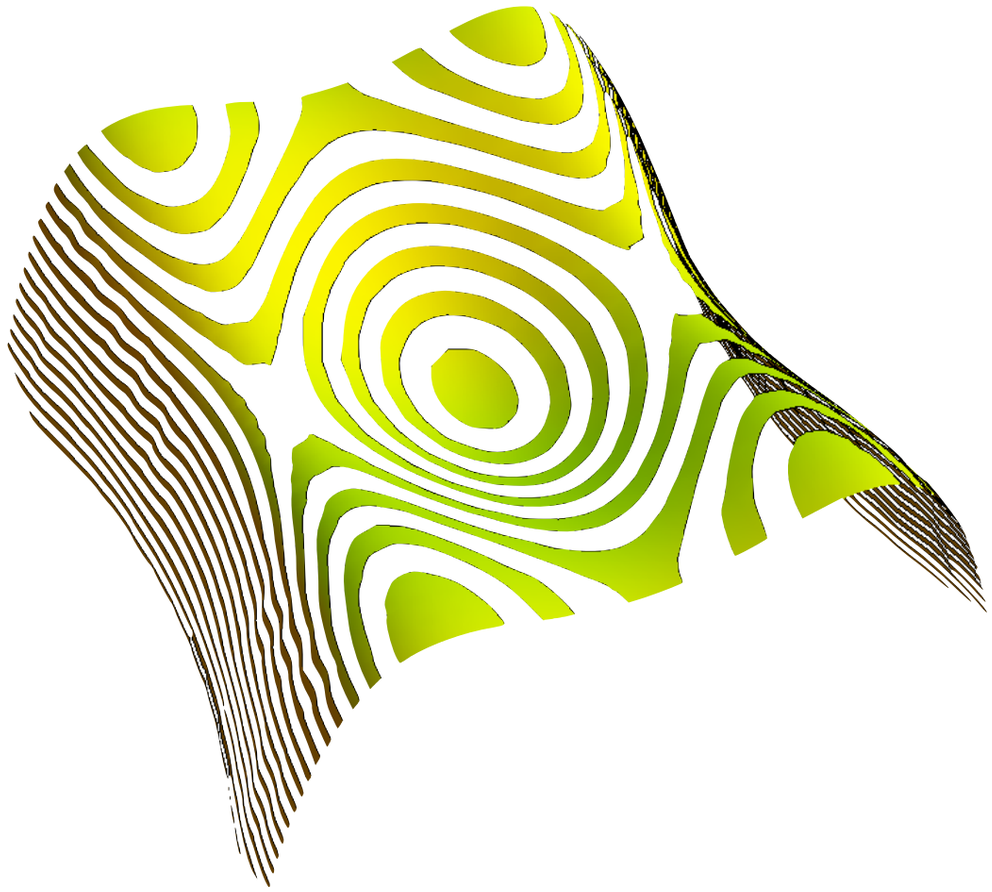
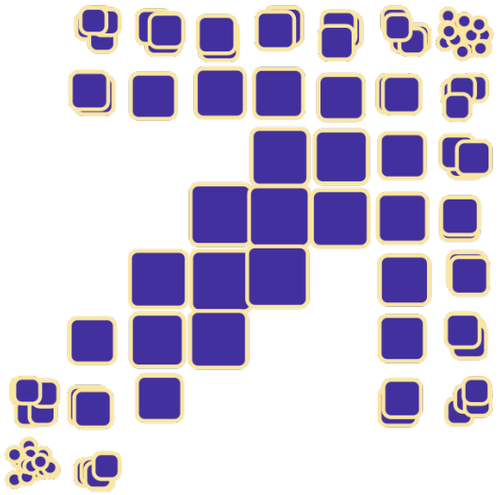




# pysparkling v0.3

*A pure Python implementation of Spark's RDD interface.*

Sven Kreiss
Hack and Tell, New York, August 26, 2015

## 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

## Input File

Read and print lines from a text file. This is **`test.py`**:

```
import sys
import pysparkling

c = pysparkling.Context()
rdd = c.textFile(sys.argv[1])

print(rdd.collect())
```

Run using:

```
$ python test.py test.py
[u'import sys', u'import pysparkling', u'', u'c =
pysparkling.Context()', u'rdd = c.textFile(sys.argv[1])', u'',
u'print(rdd.collect())', u'']
```

## Input File

Read and print lines from a text file. This is **`test.py`**:

```
import sys
import pysparkling

c = pysparkling.Context()
rdd = c.textFile(sys.argv[1])

print(rdd.collect())
```

Run using:

```
$ python test.py test.py.gz
[u'import sys', u'import pysparkling', u'', u'c =
pysparkling.Context()', u'rdd = c.textFile(sys.argv[1])', u'',
u'print(rdd.collect())', u'']
```

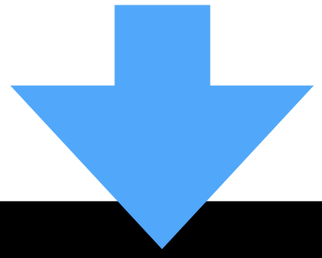
## Input File

Read and print lines from a text file. This is **`test.py`**:

```
import sys
import pysparkling

c = pysparkling.Context()
rdd = c.textFile(sys.argv[1])

print(rdd.collect())
```



Run using:

```
$ python test.py http://www.svenkreiss.com/test.py.gz
[u'import sys', u'import pysparkling', u'', u'c =
pysparkling.Context()', u'rdd = c.textFile(sys.argv[1])', u'',
u'print(rdd.collect())', u'']
```

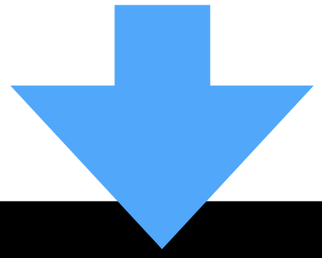
## Input File

Read and print lines from a text file. This is **`test.py`**:

```
import sys
import pysparkling

c = pysparkling.Context()
rdd = c.textFile(sys.argv[1])

print(rdd.collect())
```



Run using:

```
$ python test.py hdfs://localhost:50070/users/hadoop/test.py.bz2
[u'import sys', u'import pysparkling', u'', u'c =
pysparkling.Context()', u'rdd = c.textFile(sys.argv[1])', u'',
u'print(rdd.collect())', u'']
```

## Input File

Read and print lines from a text file. This is **`test.py`**:

```
import sys
import pysparkling

c = pysparkling.Context()
rdd = c.textFile(sys.argv[1])

print(rdd.collect())
```

Run using:

```
$ python test.py s3n://bucketname/test.py.bz2
[u'import sys', u'import pysparkling', u'', u'c =
pysparkling.Context()', u'rdd = c.textFile(sys.argv[1])', u'',
u'print(rdd.collect())', u'']
```

## Input File

Read and print lines from a text file. This is **`test.py`**:

```
import sys
import pysparkling

c = pysparkling.Context()
rdd = c.textFile(sys.argv[1])

print(rdd.collect())
```

Run using:

```
$ python test.py tes?.py.gz,s3n://bucketname/test.py
[u'import sys', u'import pysparkling', u'', u'c =
pysparkling.Context()', u'rdd = c.textFile(sys.argv[1])', u'',
u'print(rdd.collect())', u'', u'import sys', u'import pysparkling',
u'', u'c = pysparkling.Context()', u'rdd = c.textFile(sys.argv[1])',
u'', u'print(rdd.collect())', u'']
```

→ 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.

## 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.

## 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 operations based on shuffles, sorts, groups, … are still run locally. Those functions are marked in the API documentation. Again: use PySpark where appropriate.

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