edureka!

PYSPARK RDD CHEAT SHEET

Learn PySpark at www.edureka.co

PySpark RDD

Resilient Distributed Datasets (RDDs) are a distributed memory abstraction that helps a programmer to perform in-memory computations on large clusters that too in a fault-tolerant manner.

Initialization

Let's see how to start Pyspark and enter the shell

- Go to the folder where Pyspark is installed
- Run the following command

```
$ ./sbin/start-all.sh
$ spark-shell
```

Now that spark is up and running, we need to initialize spark context, which is the heart of any spark application.

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

Configurations

```
>>> from pyspark import SparkConf, SparkContext
>>> val conf = (SparkConf()
         .setMaster("local[2]")
         .setAppName("Edureka CheatSheet")
         .set("spark.executor.memory", "1g"))
>>> val sc = SparkContext(conf = conf)
```

Spark Context Inspection

Now once, spark context is initialized, it's time to check if all the versions are correct or not. We need to check the default parameters being used by SparkContext.

```
>>> sc.version                  # SparkContext Version
>>> sc.pythonVer                # Python Version
>>> sc.appName                  # Application Name
>>> sc.applicationId            # Application ID
>>> sc.master                   # Master URL
>>> str(sc.sparkHome)           # Installed Spark Path
>>> str(sc.sparkUser())         # Retreive Current SparkContext User
>>> sc.defaultParallelism       # Get default level of Parallelism
>>> sc.defaultMinPartitions     # Get minimum number of Partitions
```

Data Loading

Creating RDDs

Using Parallelized Collections

```
>>> rdd = sc.parallelize([('Jim',24),('Hope', 25),('Sue', 26)])
>>> rdd = sc.parallelize([('a',9),('b',7),('c',10)])
>>> num_rdd = sc.parallelize(range(1,5000))
```

From other RDDs

```
>>> new_rdd = rdd.groupByKey()
>>> new_rdd = rdd.map(lambda x: (x,1))
```

From a text File

```
>>> tfile_rdd = sc.textFile("/path/of_file/*.txt")
```

Reading directory of Text Files

```
>>> tfile_rdd = sc.wholeTextFiles("/path/of_directory/")
```

RDD Statistics

Maximum Value of RDD elements

```
>>> rdd.max()
```

Minimum Value of RDD elements

```
>>> rdd.min()
```

Mean value of RDD elements

```
>>> rdd.mean()
```

Standard Deviation of RDD elements

```
>>> rdd.stdev()
```

Get the Summary Statistics Count, Mean, Stdev, Max & Min

```
>>> rdd.stats()
```

Number of Partitions

```
>>> rdd.getNumPartitions()
```

Transformations and Actions

Transformations

map

```
>>> rdd = sc.parallelize(["b", "a", "c"])
>>> rdd.map(lambda x: (x, 1))
[('a', 1), ('b', 1), ('c', 1)]
```

flatMap

```
>>> rdd = sc.parallelize([2, 3, 4])
>>> rdd.flatMap(lambda x: range(1, x))
[1, 1, 1, 2, 2, 3]
```

mapPartitions

```
>>> rdd = sc.parallelize([1, 2, 3, 4], 2)
>>> def f(iterator): yield sum(iterator)
>>> rdd.mapPartitions(f).collect()
[3, 7]
```

filter

```
>>> rdd = sc.parallelize([1, 2, 3, 4, 5])
>>> rdd.filter(lambda x: x % 2 == 0).collect()
[2, 4]
```

distinct

```
>>> sorted(sc.parallelize([1, 1, 2, 3]).distinct().collect())
[1, 2, 3]
```

Actions

reduce

```
>>> from operator import add
>>> sc.parallelize([1, 2, 3, 4, 5]).reduce(add)
15
>>> sc.parallelize((2 for _ in range(10))).map(lambda x: 1)
.cache().reduce(add)
10
```

count

```
>>> sc.parallelize([2, 3, 4]).count()
3
```

first

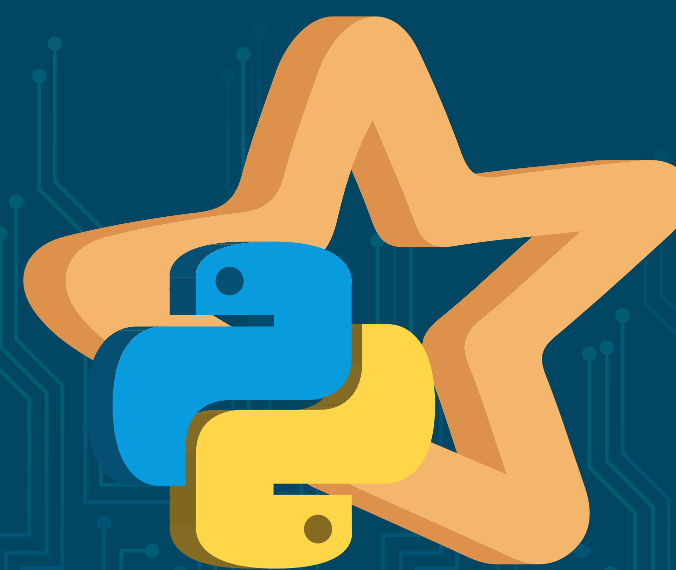
```
>>> sc.parallelize([2, 3, 4]).first()
2
```

take

```
>>> sc.parallelize([2, 3, 4, 5, 6]).cache().take(2)
[2, 3]
```

countByValue

```
>>> sorted(sc.parallelize([1, 2, 1, 2, 2], 2).countByValue().items())
[(1, 2), (2, 3)]
```



PYSPARK CERTIFICATION TRAINING

Sorting and Set Operations

Sorting and Grouping

sortBy

```
>>> tmp = [('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5)]
>>> sc.parallelize(tmp).sortBy(lambda x: x[0]).collect()
[('1', 3), ('2', 5), ('a', 1), ('b', 2), ('d', 4)]
```

sortByKey

```
>>> tmp = [('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5)]
>>> sc.parallelize(tmp).sortByKey(True, 1).collect()
[('1', 3), ('2', 5), ('a', 1), ('b', 2), ('d', 4)]
```

groupBy

```
>>> rdd = sc.parallelize([1, 1, 2, 3, 5, 8])
>>> result = rdd.groupBy(lambda x: x % 2).collect()
>>> sorted([(x, sorted(y)) for (x, y) in result])
[(0, [2, 8]), (1, [1, 1, 3, 5])]
```

groupByKey

```
>>> x = sc.parallelize([("a", 1), ("b", 1), ("a", 1)])
>>> map((lambda (x,y): (x, list(y))), sorted(x.groupByKey().collect()))
[('a', [1, 1]), ('b', [1])]
```

fold

```
>>> from operator import add
>>> sc.parallelize([1, 2, 3, 4, 5]).fold(0, add)
15
```

Set Operations

add

```
>>> rdd = sc.parallelize([1, 1, 2, 3])
>>> (rdd + rdd).collect()
[1, 1, 2, 3, 1, 1, 2, 3]
```

subtract

```
>>> x = sc.parallelize([("a", 1), ("b", 4), ("b", 5), ("a", 3)])
>>> y = sc.parallelize([("a", 3), ("c", None)])
>>> sorted(x.subtract(y).collect())
[('a', 1), ('b', 4), ('b', 5)]
```

unioin

```
>>> rdd = sc.parallelize([1, 1, 2, 3])
>>> rdd.union(rdd).collect()
[1, 1, 2, 3, 1, 1, 2, 3]
```

intersection

```
>>> rdd1 = sc.parallelize([1, 10, 2, 3, 4, 5])
>>> rdd2 = sc.parallelize([1, 6, 2, 3, 7, 8])
>>> rdd1.intersection(rdd2).collect()
[1, 2, 3]
```

cartesian

```
>>> rdd = sc.parallelize([1, 2])
>>> sorted(rdd.cartesian(rdd).collect())
[(1, 1), (1, 2), (2, 1), (2, 2)]
```

Sorting and Set Operations

saveAsTextFile

```
>>> rdd.saveAsTextFile("rdd.txt")
```

saveAsHadoopFile

```
>>> rdd.saveAsHadoopFile
("hdfs://namenodehost/parent_folder/child_folder",
'org.apache.hadoop.mapred.TextOutputFormat')
```

saveAsPickleFile

```
>>> tmpFile = NamedTemporaryFile(delete=True)
>>> tmpFile.close()
>>> sc.parallelize([1, 2, 'spark', 'rdd'])
.saveAsPickleFile(tmpFile.name, 3)
>>> sorted(sc.pickleFile(tmpFile.name, 5).collect())
[1, 2, 'rdd', 'spark']
```

Stopping SparkContext and Spark Daemons

Stopping SparkContext

```
>>> sc.stop()
```

Stopping Spark Daemons

```
$ ./sbin/stop-all.sh
```