databrickssolutionsRDDsTutorialDatabricks

```
print( spark.version, sc.appName, sc.master )
```

```
2.4.5 Databricks Shell local[8]
```

```
%fs
ls /FileStore/tables/100M.data
```

path
dbfs:/FileStore/tables/100M.data

```
f100 = sc.textFile("/FileStore/tables/100M.data")
f100.take(2)
#f100.saveAsTextFile("/FileStore/tables/f100")
f100.coalesce(1).saveAsTextFile("/FileStore/tables/f100_2")
```

```
Out[10]: ['twinkle twinkle little star', 'how I wonder what you are']
```

```
%fs
ls /FileStore/tables/f100
```

path
dbfs:/FileStore/tables/f100/_SUCCESS
dbfs:/FileStore/tables/f100/part-00000
dbfs:/FileStore/tables/f100/part-00001
dbfs:/FileStore/tables/f100/part-00002
dbfs:/FileStore/tables/f100/part-00003
dbfs:/FileStore/tables/f100/part-00004

```
print (f100.getNumPartitions())
f100_3 = f100.repartition( 4*sc.defaultParallelism )
print( f100_3.getNumPartitions() )
```

```
2
32
```

RDDs: union, intersection, distinct and subtract

```
u1 = ["m1","m2","m3"]
u2 = ["m2","m3","m4","m5"]

u1 = sc.parallelize( u1 )
u2 = sc.parallelize( u2 )

print ("<> Meetings in common: ", u1.intersection(u2).collect())
print ("<> Meetings attended by either of users (duplicates): ",
u1.union(u2).collect())
print ("<> Meetings attended by either of users: ",
u1.union(u2).distinct().collect())
print ("<> Meetings only for user1: ", u1.subtract(u2).collect())
print ("<> List of reccommendations:", [u1.subtract(u2).collect(),
u2.subtract(u1).collect()])
```

```
<> Meetings in common:  ['m2', 'm3']
<> Meetings attended by either of users (duplicates):  ['m1', 'm2', 'm3', 'm2', 'm3
', 'm4', 'm5']
<> Meetings attended by either of users:  ['m4', 'm5', 'm2', 'm1', 'm3']
<> Meetings only for user1:  ['m1']
<> List of reccommendations: [['m1'], ['m4', 'm5']]
```

RDDs: map

```
path = "/FileStore/tables/"
f = path + "stars.txt"
stars = sc.textFile(f)
print( "<> stars type", type(stars) )
print ( "<> First line of stars: ", stars.take(2) )
stars2 = stars.map( lambda line: line.upper() )
print ( "<> First line of stars2: ", stars2.take(1) )

print ( "<> Type of stars2: ", type(stars2) )
```

```
<> stars type <class 'pyspark.rdd.RDD'>
<> First line of stars:  ['A star is type of astronomical object consisting of a lu
minous spheroid of plasma held', 'together by its own gravity. The nearest star to
Earth is the Sun. Many other stars are']
<> First line of stars2:  ['A STAR IS TYPE OF ASTRONOMICAL OBJECT CONSISTING OF A L
UMINOUS SPHEROID OF PLASMA HELD']
<> Type of stars2:  <class 'pyspark.rdd.PipelinedRDD'>
```

```
def makeUpper( elem ) :
    return elem.upper()

stars.map( makeUpper ).take(1)
```

```
Out[8]: ['A STAR IS TYPE OF ASTRONOMICAL OBJECT CONSISTING OF A LUMINOUS SPHEROID O
F PLASMA HELD']
```

```
stars.take(2)
```

```
Out[21]: ['A star is type of astronomical object consisting of a luminous spheroid
of plasma held',
 'together by its own gravity. The nearest star to Earth is the Sun. Many other sta
rs are']
```

RDDs: Pars a text file (splitting)

```
stars.map(lambda ll: ll.split(" ")).take(2)
```

```
Out[16]: [['A',
  'star',
  'is',
  'type',
  'of',
  'astronomical',
  'object',
  'consisting',
  'of',
  'a',
  'luminous',
  'spheroid',
  'of',
  'plasma',
  'held'],
 ['together',
  'by',
  'its',
  'own',
  'gravity.',
  'The',
```

RDDs: flatMap

```
stars.flatMap(lambda ll: ll.split(" ")).take(2)
```

```
Out[56]: ['A', 'star']
```

RDDs: map, filter and reduce

```
xrdd = sc.parallelize(range(-2,3))
print ( "<> map: ", xrdd.map(lambda f: f<0).collect() )
print ( "<> filter: ", xrdd.filter(lambda f: f<0).collect() )
print ( "<> reduce: ", xrdd.reduce(lambda f,g: f+g) )

<> map:  [True, True, False, False, False]
<> filter:  [-2, -1]
<> reduce:  0
```

RDDs: Pair RDD from a list of tuples :

```
tuplist = [("try 1"), ("it 2"), ("hard 3")]
regularRdd = sc.parallelize(tuplist)
print ( "<> regularRdd content: ", regularRdd.collect() )
print ( "<> regularRDD type: ", type(regularRdd) )

pairRdd = regularRdd.map( lambda ll: ( ll.split(" ")[0], int(ll.split(" ")[1]) ) )
print ( "<> pairRdd content: ", pairRdd.collect() )
print ( "<> pairRdd type: ", type(pairRdd) )

<> regularRdd content:  ['try 1', 'it 2', 'hard 3']
<> regularRDD type:  <class 'pyspark.rdd.RDD'>
<> pairRdd content:  [('try', 1), ('it', 2), ('hard', 3)]
<> pairRdd type:  <class 'pyspark.rdd.PipelinedRDD'>
```

RDDs: Pair RDD from zipping 2 regular RDDs

```
rdd1 = sc.parallelize(["try", "it", "hard"])
rdd2 = sc.parallelize([1, 2, 3])
pairRdd = rdd1.zip(rdd2)
print ( "<> pairRdd content: ", pairRdd.collect() )

<> pairRdd content:  [('try', 1), ('it', 2), ('hard', 3)]
```

Pair RDD from keyBy

```
rdd = sc.parallelize([ "gate 9995 zeppelin", "gate 8888 jupyter", "gate 4040 spark"
])
pairRdd = rdd.keyBy( lambda e : e.split(" ")[1] )
pairRdd.collect()

Out[18]: [('9995', 'gate 9995 zeppelin'),
 ('8888', 'gate 8888 jupyter'),
```

```
 ('4040', 'gate 4040 spark')]
```

RDDs: mapValues

```
print ( "<> mapValues: ", pairRdd.mapValues(lambda f: f*f).collect() )
print ( "<> map: ", pairRdd.map(lambda f: f*f).collect() )

<> mapValues:  [('try', 1), ('it', 4), ('hard', 9)]
<> map:  [1, 4, 9]
```

RDDs: reduceByKey

```
from operator import add
rdd = sc.parallelize( ["A fool thinks himself to be wise but a wise man knows
himself to be a fool"] )
rdd.map( lambda ll : ll.lower() ).flatMap( lambda w: w.split(" ") ).map( lambda w:
(w,1) ).reduceByKey( add ).sortBy( lambda w: w[1], ascending=False ).collect()

Out[10]: [('a', 3),
 ('fool', 2),
 ('to', 2),
 ('himself', 2),
 ('be', 2),
 ('wise', 2),
 ('but', 1),
 ('knows', 1),
 ('man', 1),
 ('thinks', 1)]

a = rdd.flatMap(lambda w: w.split(" ")).map(lambda w: (w,1))
b = a.reduceByKey(lambda x,y: x+y)
b.sortBy(lambda w: w[1]).collect()

Out[62]: [('but', 1),
 ('thinks', 1),
 ('A', 1),
 ('knows', 1),
 ('man', 1),
 ('fool', 2),
 ('to', 2),
 ('himself', 2),
 ('a', 2),
 ('be', 2),
 ('wise', 2)]
```

Exercise : Simple RDD manip

```
ls = list((1,2,3,4,500,6,7,8,9,10,11,12,100))
numberRdd = sc.parallelize(ls)

print( "<> Statistics", numberRdd.stats() , "\n")

month1Rdd =
sc.parallelize(['janvier','fevrier','mars','avril','mai','juin','juillet','aout'])
month2Rdd = sc.parallelize(['septembre','octobre','novembre','decembre'])
monthRdd = month1Rdd.union(month2Rdd)
print( "<> monthRdd :", monthRdd.collect(), "\n")

lrdd1 =  sc.parallelize(range(1,9),month1Rdd.getNumPartitions())
lrdd2 =  sc.parallelize(range(9,13),month2Rdd.getNumPartitions())
keyInd = lrdd1.union(lrdd2)
print(( '<> Key indices: {} '.format(keyInd.collect()) ), "\n")

print ( '<> Number of partitions for monthRDD and keyInd:{},
{}'.format(monthRdd.getNumPartitions(), keyInd.getNumPartitions()), "\n" )

monthWithIndexRdd = keyInd.zip(monthRdd)

print ( "<> Pair RDD :", monthWithIndexRdd.collect(), "\n" )

print( "<> Using zipWithIndex method : ", monthRdd.zipWithIndex().collect(), "\n" )

p1 = monthWithIndexRdd.map(lambda tup:  (tup[0]+1,tup[1]))
print ( "<> Add 1 to all keys : ", p1.collect())
```

```
<> Statistics (count: 13, mean: 51.76923076923077, stdev: 131.787797875, max: 500.0
, min: 1.0)

<> monthRdd : ['janvier', 'fevrier', 'mars', 'avril', 'mai', 'juin', 'juillet', 'ao
ut', 'septembre', 'octobre', 'novembre', 'decembre']

<> Key indices: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

<> Number of partitions for monthRDD and keyInd:16,16

<> Pair RDD : [(1, 'janvier'), (2, 'fevrier'), (3, 'mars'), (4, 'avril'), (5, 'mai'
), (6, 'juin'), (7, 'juillet'), (8, 'aout'), (9, 'septembre'), (10, 'octobre'), (11
, 'novembre'), (12, 'decembre')]

<> Using zipWithIndex method :  [('janvier', 0), ('fevrier', 1), ('mars', 2), ('avr
il', 3), ('mai', 4), ('juin', 5), ('juillet', 6), ('aout', 7), ('septembre', 8), ('
octobre', 9), ('novembre', 10), ('decembre', 11)]
```

```
<> Add 1 to all keys :  [(2, 'janvier'), (3, 'fevrier'), (4, 'mars'), (5, 'avril'),
(6, 'mai'), (7, 'juin'), (8, 'juillet'), (9, 'aout'), (10, 'septembre'), (11, 'octo
bre'), (12, 'novembre'), (13, 'decembre')]
```

Exercise : Prenoms

```
import re

fname = path + "Prenoms.csv"
#prenomsRdd = sc.textFile(fname)\
#        .map(lambda ll: re.sub(r'\\','-',ll)).map(lambda ll: ll.split(';'))

prenomsRdd = sc.textFile(fname).map(lambda ll: ll.split(';'))



print('prenomsRdd: ')
print(prenomsRdd.count())
print(prenomsRdd.take(5))

sexeRdd = prenomsRdd.map(lambda ls: ls[1])
fRdd = sexeRdd.filter(lambda ls: ls[0]=='f')
mRdd = sexeRdd.filter(lambda ls: ls[0]=='m')
print('<> Number of female names:{}'.format(fRdd.count()))
print('<> Number of male names:{}'.format(mRdd.count()))

distinctSexeRdd = sexeRdd.distinct()

langage1Rdd = prenomsRdd.map(lambda ls: ls[2])
langage2Rdd = langage1Rdd.filter(lambda el: len(el.split(','))==1 ).filter(lambda
el: el!='' )
langagePairRdd = langage2Rdd.map(lambda el: (el,1))

numLangage = langagePairRdd.reduceByKey(lambda x,y: x+y)
print('<> Top five languages:{}'.format(numLangage.sortBy(lambda tup:
tup[1],ascending=False).take(5)))
print('<> Least spoken languages: {}'.format(numLangage.sortBy(lambda tup:
tup[1]).take(5)))



prenomsRdd:
11627
[['aaliyah', 'f', 'english (modern)', '0'], ['aapeli', 'm', 'finnish', '0'], ['aapo
', 'm', 'finnish', '0'], ['aaren', 'm,f', 'english', '0'], ['aarne', 'm', 'finnish'
, '0']]
<> Number of female names:5460
<> Number of male names:6167
```

```
<> Top five languages:[('english', 2659), ('arabic', 457), ('italian', 436), ('fren
ch', 412), ('irish', 371)]
<> Least spoken languages: [('medieval english', 1), ('macedonian', 1), ('celtic my
thology (latinized)', 2), ('proven?al', 2), ('biblical (original)', 2)]
```

RDDs: An example for frequency calculation

```
rdd = sc.parallelize( [
    "A beautiful woman delights the eye ; a wise woman , the understanding ; a pure
woman , the soul"
] )
wordNum =  rdd.flatMap(lambda w: w.split(" ")).count()
print ( "<> Number of words: ", wordNum )

rdd.flatMap(lambda line: line.lower().split(" ")).map(lambda w:
(w,1)).reduceByKey(lambda a,b: a+b)\
    .mapValues(lambda s: float(s)/wordNum).collect()
```

```
<> Number of words:  20
Out[33]: [('a', 0.15),
 ('beautiful', 0.05),
 ('pure', 0.05),
 ('woman', 0.15),
 ('eye', 0.05),
 ('understanding', 0.05),
 ('delights', 0.05),
 (';', 0.1),
 ('the', 0.15),
 ('wise', 0.05),
 (',', 0.1),
 ('soul', 0.05)]
```

Exercise : Stars

```
rddt = sc.textFile( "/FileStore/tables/stars.txt" )
rddt2 = rddt.flatMap(lambda line : line.lower().split(" ") ).\
  map( lambda w : (w, 1) ).\
  reduceByKey( lambda a, b : a+b )

print( rddt2.min( lambda e : e[1] ), rddt2.max( lambda e : e[1] ) )

rddt2.sortBy(lambda e : e[1], ascending = False).collect()
```

```
('(chemical', 1) ('the', 21)
Out[30]: [('the', 21),
 ('of', 15),
 ('a', 13),
```

```
 ('and', 10),
 ('its', 9),
 ('star', 9),
 ('stars', 7),
 ('to', 6),
 ('that', 5),
 ('by', 4),
 ('is', 3),
 ('life,', 3),
 ('other', 3),
 ('in', 3),
 ('into', 3),
 ('are', 3),
 ("star's", 3),
 ('from', 3),
 ('many', 3),
```

Exercise : Paris Election

```
 1. Opening /FileStore/tables/resultats_electoraux.csv ...
Nombre des ligne:179336
 La premirer ligne:L??gislatives 2012 - 1er Tour;06/10/2012;PARIS 04;1353;831;MAJDA
;Jacky;27
 2. Making rsltElect2Rdd in json format ...
 3. Saving the new RDD as rsltElect2Rdd.txt ...
 4. Extracting the 2017-presidential info ...
 Number of votes gained by the candidates in Paris region:
[('MACRON_Emmanuel', 375005), ('FILLON_Fran??ois', 284744), ('M??LENCHON_Jean-Luc',
210548), ('HAMON_Beno??t', 109550), ('LE PEN_Marine', 53719), ('DUPONT-AIGNAN_Nicol
as', 17997), ('ASSELINEAU_Fran??ois', 8335), ('POUTOU_Philippe', 6799), ('LASSALLE_
Jean', 5490), ('ARTHAUD_Nathalie', 2897), ('CHEMINADE_Jacques', 1472)]
```