**Closure** :

A closure is a function, whose return value depends on the value of one or more variables declared outside this function.

Snippet:

case class Field(…..)

// Funtion declaration 1

def convertToField (line : String) : Field {…..} ///Return Field object

// Funtion declaration 2

val convertToField2 = (line : String) => {….. return Field(….)}

///Return Field object

rdd.map(convertToField)

Here return of map depends of return of ‘convertToField’ method

**Curring:**

Technique of transforming a function that takes multiple arguments into a function that takes a single argument

Snippet Before currying:

def add(x:Int, y:Int) = x + y

add(1, 2) // 3

add(7, 3) // 10

Snippet after currying:

def add(x:Int) = (y:Int) => x + y

add(1)(2) // 3

add(7)(3) // 10

RDD:

Basic RDD – Each element is a single object. Like RDD<Int>, RDD<String> or RDD<Custom Class>

Paired RDD – Each element is a key/Value pair. Every record is treated as a tuple

**map function:**

‘map’ function applied the function definition passed on to it against each entry of rdd. It is a Transformation. Function is available for All types of RDDs.

Snippet:

Let Rdd be of type RDD<Int>

rdd.map(\_.split(“,”))

‘\_.split(“,”)’ short hand to - split each entry of rdd (‘\_’ is a place holder)mapped by ‘map function)

NOTE: ‘map’ function support closure.

**reduce function**

‘reduce’ function return a single output after traversing through rdd. It is an Action. Function is available for All types of RDDs.

Snippet:

Let Rdd be of type RDD<Int>

rdd.reduce((v1, v2) => v1+v2 )

In the above example ‘reduce’ function take first entry of int RDD ‘v1’ add it to second entry ‘v2’ and so on so forth till the all the entries are traversed. The result of this snippet will be sum of all elements in rdd

**‘aggregate’ function**

‘aggregate’ function allows to perform more than one operation on RDD in parallel. It is an Action. Function is available for All types of RDDs.

Snippet:

Let Rdd be of type RDD<Float>;

rdd.aggregate (0.0,0)((rec, value) =>rec.\_1+ value, rec.\_2+1, (rec1, rec2) => rec1.\_1 +rec2.\_1, rec1.\_2 +rec2.\_2 )

‘aggregate’ function support : curring technique.

In the above example aggregate’ return a tuple when applied on RDD<Float>.

(0.0,0) : initial value. First element is initial value for sum (hence float) and second for count of elements traversed

(rec, value) =>rec.\_1+ value, rec.\_2+1 : Short hand for adding value which is returned from RDD of type float to first entry in tuple for maintaining sum and incrementing second entry in tuple kept for count

(rec1, rec2) => rec1.\_1 +rec2.\_1, rec1.\_2 +rec2.\_2 : Short hand for adding results from different partitions; which computes sum and count in their respective partition with earlier method

**countByValue funtion**

countByValue() is an Action that returns the Map of each unique value with its count. Function is available for All types of RDDs.

Snippet:

scala> val inputrdd = sc.parallelize{ Seq(10, 4, 3, 3) }

inputrdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[28] at parallelize at :47

scala> inputrdd.countByValue()

res34: scala.collection.Map[Int,Long] = Map(10 -> 1, 3 -> 2, 4 -> 1)

**‘keys’ function:**

‘keys’ function return all keys in a Paired RDD. Function is available only for Paired RDD. It is a Transformation.

**‘values’ function:**

‘values’ function return all values in a Paired RDD. Function is available only for Paired RDD. It is a Transformation.

**‘mapValues’ function:**

‘mapValues’ function – map input function against all values but not keys. Function is available only for Paired RDD. It is a Transformation.

Snippet:

pairedrdd.mapValues(\_.split(“,”))

All values in Paired RDD will be split with ‘,’ with about snippet

**‘groupByKey’ function**

‘groupByKey’ function groups all values against same key within same RDD and return (key, itereator of list of values grouped). It is a Transformation. Function is available only for Paired RDD.

**‘cogroup’ function**

‘cogroup’ function groups values group values against same key across multiple RDDs. It is a Transformation. Function is available only for Paired RDD.

**‘reducebyKey’ function**

‘reducebyKey’ function preform reduce action against all values which has same key. It is a Transformation. Function is available only for Paired RDD.

**‘combineByKey’ function**

‘combineByKey’ function is analogous to aggregate function, but it acts only on values which has same key. It is a Transformation. Function is available only for Paired RDD.

combineByKey(createCombinerFunction, mergeFunction,mergeCombinerFunction)

createCombinerFunction: this initializes a values when a key is first seen within a partition

mergeFunction: Specifies how values with the same key should be combined within a partition

mergeCombinerFunction: Decides how putputs from different partitions has to be combined

**‘join’ functions**

‘join’ function joins two paired rdds against keys and return values for key available in both RDDs. Function is available only for Paired RDD. It is a Transformation

**‘leftOuterJoin’ functions**

‘leftOuterJoin’ function joins two paired rdds against keys and return values for keys available in both RDD and keys only present in first RDD . Function is available only for Paired RDD. It is a Transformation

**‘rightOuterJoin’ functions**

‘rightOuterJoin’ function joins two paired rdds against keys and return values for keys available in both RDD and keys only present in first RDD . Function is available only for Paired RDD. It is a Transformation

**‘countByKey’ function**

‘countByKey’ function return count of values for each key. It is a Action. Function is available only for Paired RDD.

**‘lookup’ function**

‘lookup’ function return values for given key. It is a Action. Function is available only for Paired RDD.

**‘collectAsMap’ function**

‘collectAsMap’ function return all key-value pair. It is a Action. Function is available only for Paired RDD.

**‘sortBy’ function**

‘sortBy’ function is a Transformation. Available for all RDDs.

Snippet:

pairedRDD.sortby(\_.\_2)

// Sort pairedRDD based on second element of tuple which is value in ascending order.

pairedRDD.sortby( - \_.\_2)

// Sort pairedRDD based on second element of tuple which is value in descending order marked by the sign ‘-‘.