Experiment no. 10

In-built Functions

Map Function, Filer Function, Count Function ,Distinct Function ,Union Function ,Intersection Function, Cartesian Function ,sortByKey Function ,groupByKey Function ,reducedByKey Function ,Co-Group Function ,First Function ,Take Function

Prof. Dhanashree

Spark Map function

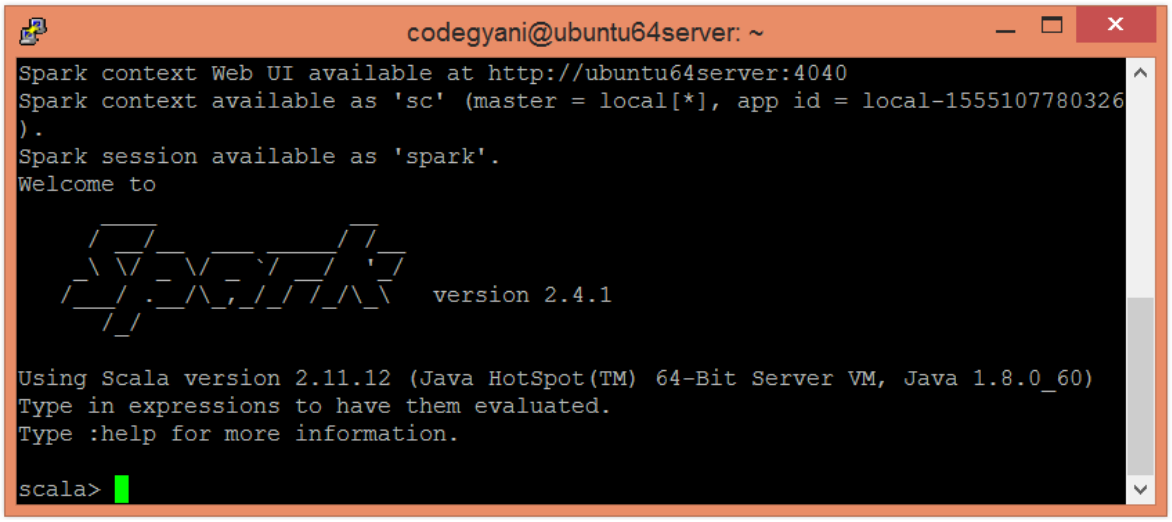
In Spark, the Map passes each element of the source through a function and forms a new distributed dataset.

Example of Map function

In this example, we add a constant value 10 to each element.

- **To open the spark in Scala mode, follow the below command**

1. **$ spark-shell**



- **Create an RDD using parallelized collection.**

1. **scala> val data = sc.parallelize(List(10,20,30))**
    - **Now, we can read the generated result by using the following command.**

1. **scala> data.collect**

    - **Apply the map function and pass the expression required to perform.**

1. **scala> val mapfunc = data.map(x => x+10)**
    - **Now, we can read the generated result by using the following command.**

1. **scala> mapfunc.collect**

```
codegyani@ubuntu64server: ~

scala> val mapfunc = data.map(x => x+10)
mapfunc: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[2] at map at <console>
:25

scala> mapfunc.collect
res3: Array[Int] = Array(20, 30, 40)

scala>
```

Here, we got the desired output.

Spark Filter Function

In Spark, the Filter function returns a new dataset formed by selecting those elements of the source on returns true. So, it retrieves only the elements that satisfy the given condition.

Example of Filter function

In this example, we filter the given data and retrieve all the values except 35.

- **To open the spark in Scala mode, follow the below command.**

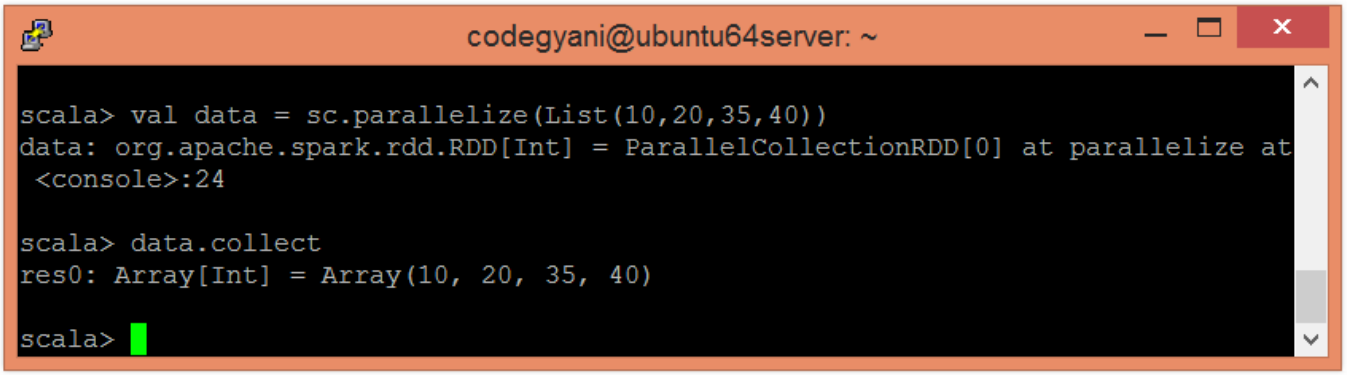
1. **$ spark-shell**

```
codegyani@ubuntu64server: ~
Spark context Web UI available at http://ubuntu64server:4040
Spark context available as 'sc' (master = local[*], app id = local-1555107780326
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_60)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

- **Create an RDD using parallelized collection.**

1. **scala> val data = sc.parallelize(List(10,20,35,40))**
   - **Now, we can read the generated result by using the following command.**

1. **scala> data.collect**



   - **Apply filter function and pass the expression required to perform.**

1. **scala> val filterfunc = data.filter(x => x!=35)**
   - **Now, we can read the generated result by using the following command.**

1. **scala> filterfunc.collect**

```
scala> val filterfunc = data.filter(x => x!=35)
filterfunc: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[2] at filter at <co
nsole>:25

scala> filterfunc.collect
res2: Array[Int] = Array(10, 20, 40)

scala>
```

Here, we got the desired output.

Spark Count Function

In Spark, the Count function returns the number of elements present in the dataset.

Example of Count function

In this example, we count the number of elements exist in the dataset.

- **Create an RDD using parallelized collection.**

1. **scala> val data = sc.parallelize(List(1,2,3,4,5))**

- **Now, we can read the generated result by using the following command.**

1. **scala> data.collect**

```
scala> val data = sc.parallelize(List(1,2,3,4,5))
data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[7] at parallelize at
 <console>:24

scala> data.collect
res5: Array[Int] = Array(1, 2, 3, 4, 5)

scala>
```

- **Apply count() function to count number of elements.**

1. **scala> val countfunc = data.count()**

```
scala> val countfunc = data.count()
countfunc: Long = 5
```

Here, we got the desired output.

Spark Distinct Function

In Spark, the Distinct function returns the distinct elements from the provided dataset.

Example of Distinct function

In this example, we ignore the duplicate elements and retrieves only the distinct elements.

- **To open the spark in Scala mode, follow the below command.**

1. **$ spark-shell**

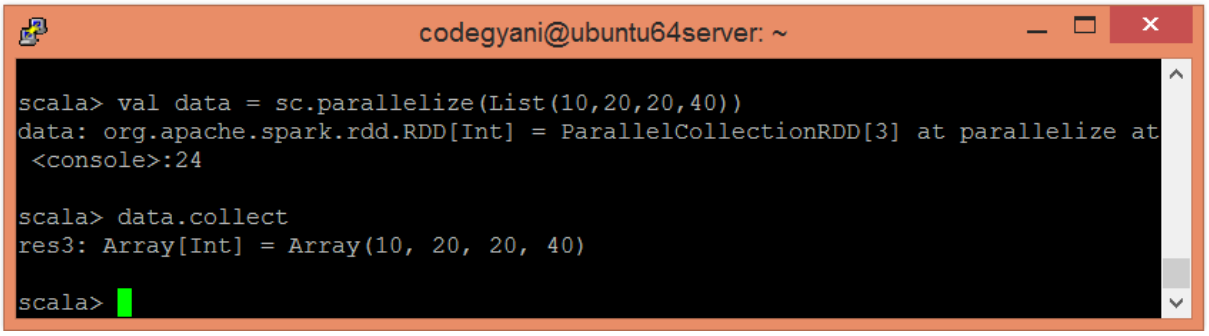
```
Spark context Web UI available at http://ubuntu64server:4040
Spark context available as 'sc' (master = local[*], app id = local-1555
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

- **Create an RDD using parallelized collection.**

1. **scala> val data = sc.parallelize(List(10,20,20,40))**

- **Now, we can read the generated result by using the following command.**

1. **scala> data.collect**

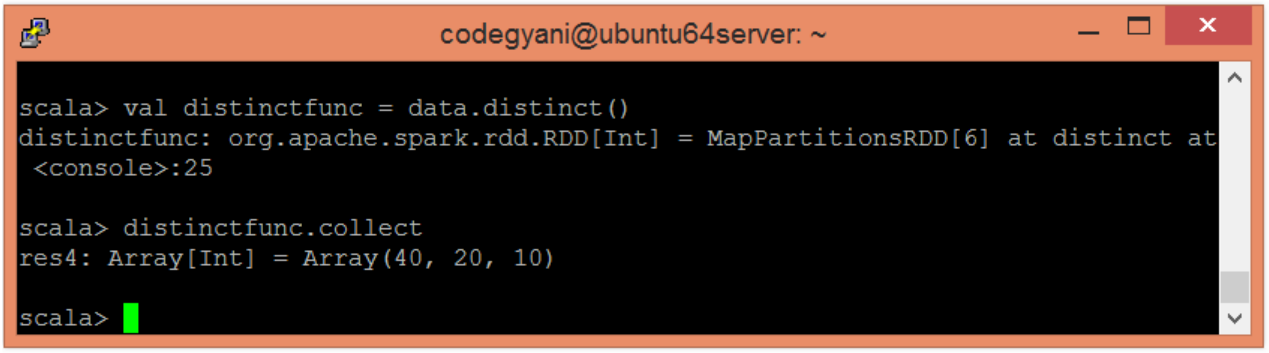


- **Apply distinct() function to ignore duplicate elements.**

1. **scala> val distinctfunc = data.distinct()**

- **Now, we can read the generated result by using the following command.**

1. **scala> distinctfunc.collect**



Here, we got the desired output.

Spark Union Function

In Spark, Union function returns a new dataset that contains the combination of elements prese datasets.

Example of Union function

In this example, we combine the elements of two datasets.

- **To open the spark in Scala mode, follow the below command.**

1. **$ spark-shell**

```
codegyani@ubuntu64server: ~
Spark context Web UI available at http://ubuntu64server:4040
Spark context available as 'sc' (master = local[*], app id = local-1555
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

- **Create an RDD using parallelized collection.**

1. **scala> val data1 = sc.parallelize(List(1,2))**

- **Now, we can read the generated result by using the following command.**

1. **scala> data1.collect**

```
codegyani@ubuntu64server: ~

scala> val data1 = sc.parallelize(List(1,2))
data1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[8] at parallelize a
t <console>:24

scala> data1.collect
res7: Array[Int] = Array(1, 2)

scala>
```

- **Create another RDD using parallelized collection.**

1. **scala> val data2 = sc.parallelize(List(3,4,5))**

- **Now, we can read the generated result by using the following command.**

1. **scala> data2.collect**

```
scala> val data2 = sc.parallelize(List(3,4,5))
data2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[9] at parallelize a
t <console>:24

scala> data2.collect
res8: Array[Int] = Array(3, 4, 5)

scala>
```

- **Apply union() function to return the union of the elements.**

1. **scala> val unionfunc = data1.union(data2)**

- **Now, we can read the generated result by using the following command.**

1. **scala> unionfunc.collect**

```
scala> val unionfunc = data1.union(data2)
unionfunc: org.apache.spark.rdd.RDD[Int] = UnionRDD[10] at union at <console>:27

scala> unionfunc.collect
res9: Array[Int] = Array(1, 2, 3, 4, 5)

scala>
```

Here, we got the desired output

Spark Intersection Function

In Spark, Intersection function returns a new dataset that contains the intersection of elements prese datasets. So, it returns only a single row. This function behaves just like the INTERSECT query in SQL.

Example of Intersection function

In this example, we intersect the elements of two datasets.

- **To open the Spark in Scala mode, follow the below command.**

1. **$ spark-shell**

```
codegyani@ubuntu64server: ~
Spark context Web UI available at http://ubuntu64server:4040
Spark context available as 'sc' (master = local[*], app id = local-1555
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

- **Create an RDD using the parallelized collection.**

1. **scala> val data1 = sc.parallelize(List(1,2,3))**
   - **Now, we can read the generated result by using the following command.**

1. **scala> data1.collect**

```
codegyani@ubuntu64server: ~

scala> val data1 = sc.parallelize(List(1,2,3))
data1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[3] at para
t <console>:24

scala> data1.collect
res3: Array[Int] = Array(1, 2, 3)

scala>
```

- **Create another RDD using parallelized collection.**

1. **scala> val data2 = sc.parallelize(List(3,4,5))**
   - **Now, we can read the generated result by using the following command.**

1. **scala> data2.collect**

```
scala> val data2 = sc.parallelize(List(3,4,5))
data2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[4] at parallelize a
t <console>:24

scala> data2.collect
res4: Array[Int] = Array(3, 4, 5)

scala>
```

- **Apply intersection() function to return the intersection of the elements.**

1. **scala> val intersectfunc = data1.intersection(data2)**
   - **Now, we can read the generated result by using the following command.**

1. **scala> intersectfunc.collect**

```
scala> val intersectfunc = data1.intersection(data2)
intersectfunc: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[10] at intersect
ion at <console>:27

scala> intersectfunc.collect
res5: Array[Int] = Array(3)

scala>
```

Here, we got the desired output.

Spark Cartesian Function

In Spark, the Cartesian function generates a Cartesian product of two datasets and returns all the pos of pairs. Here, each element of one dataset is paired with each element of another dataset.

Example of Cartesian function

In this example, we generate a Cartesian product of two datasets.

- **To open the Spark in Scala mode, follow the below command.**

1. **$ spark-shell**

```
codegyani@ubuntu64server: ~
Spark context Web UI available at http://ubuntu64server:4040
Spark context available as 'sc' (master = local[*], app id = local-1555
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

- **Create an RDD using the parallelized collection.**

1. **scala> val data1 = sc.parallelize(List(1,2,3))**
   - **Now, we can read the generated result by using the following command.**

1. **scala> data1.collect**

```
codegyani@ubuntu64server: ~

scala> val data1 = sc.parallelize(List(1,2,3))
data1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[11] at par
at <console>:24

scala> data1.collect
res6: Array[Int] = Array(1, 2, 3)

scala>
```

- **Create another RDD using the parallelized collection.**

1. **scala> val data2 = sc.parallelize(List(3,4,5))**
   - **Now, we can read the generated result by using the following command.**

1. **scala> data2.collect**

```
scala> val data2 = sc.parallelize(List(3,4,5))
data2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[12] at parallelize
at <console>:24

scala> data2.collect
res7: Array[Int] = Array(3, 4, 5)

scala>
```

- **Apply cartesian() function to return the Cartesian product of the elements.**

1. **scala> val cartesianfunc = data1.cartesian(data2)**
   - **Now, we can read the generated result by using the following command.**

1. **scala> cartesianfunc.collect**

```
scala> val cartesianfunc = data1.cartesian(data2)
cartesianfunc: org.apache.spark.rdd.RDD[(Int, Int)] = CartesianRDD[13] at cartes
ian at <console>:27

scala> cartesianfunc.collect
res8: Array[(Int, Int)] = Array((1,3), (1,4), (1,5), (2,3), (2,4), (2,5), (3,3),
 (3,4), (3,5))

scala>
```

Here, we got the desired output.

Spark sortByKey Function

In Spark, the sortByKey function maintains the order of elements. It receives key-value pairs (K, V) as elements in ascending or descending order and generates a dataset in an order.

Example of sortByKey Function

In this example, we arrange the elements of dataset in ascending and descending order.

- **To open the Spark in Scala mode, follow the below command.**

1. **$ spark-shell**

```
codegyani@ubuntu64server: ~
Spark context Web UI available at http://ubuntu64server:4040
Spark context available as 'sc' (master = local[*], app id = local-1555107780326
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_60)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

- **Create an RDD using the parallelized collection.**

1. **scala> val data = sc.parallelize(Seq(("C",3),("A",1),("D",4),("B",2),("E",5)))**

Now, we can read the generated result by using the following command.

1. **scala> data.collect**

```
codegyani@ubuntu64server: ~
scala> val data = sc.parallelize(Seq(("C",3),("A",1),("D",4),("B",2),("E",5)))
data: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[0] at para
llelize at <console>:24

scala> data.collect
res0: Array[(String, Int)] = Array((C,3), (A,1), (D,4), (B,2), (E,5))

scala>
```

For ascending,

- **Apply sortByKey() function to ignore duplicate elements.**

1. **scala> val sortfunc = data.sortByKey()**
   - **Now, we can read the generated result by using the following command.**

1. **scala> sortfunc.collect**

```
codegyani@ubuntu64server: ~

scala> val sortfunc = data.sortByKey()
sortfunc: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[1] at sortByKey
at <console>:25

scala> sortfunc.collect
res1: Array[(String, Int)] = Array((A,1), (B,2), (C,3), (D,4), (E,5))

scala>
```

Here, we got the desired output.

For descending,

- **Apply sortByKey() function and pass Boolean type as parameter.**

1. **scala> val sortfunc = data.sortByKey(false)**
   - **Now, we can read the generated result by using the following command.**

1. **scala> sortfunc.collect**

```
codegyani@ubuntu64server: ~

scala> val sortfunc = data.sortByKey(false)
sortfunc: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[2] at sortByKey
at <console>:25

scala> sortfunc.collect
res2: Array[(String, Int)] = Array((E,5), (D,4), (C,3), (B,2), (A,1))

scala>
```

Here, we got the desired output.

Spark groupByKey Function

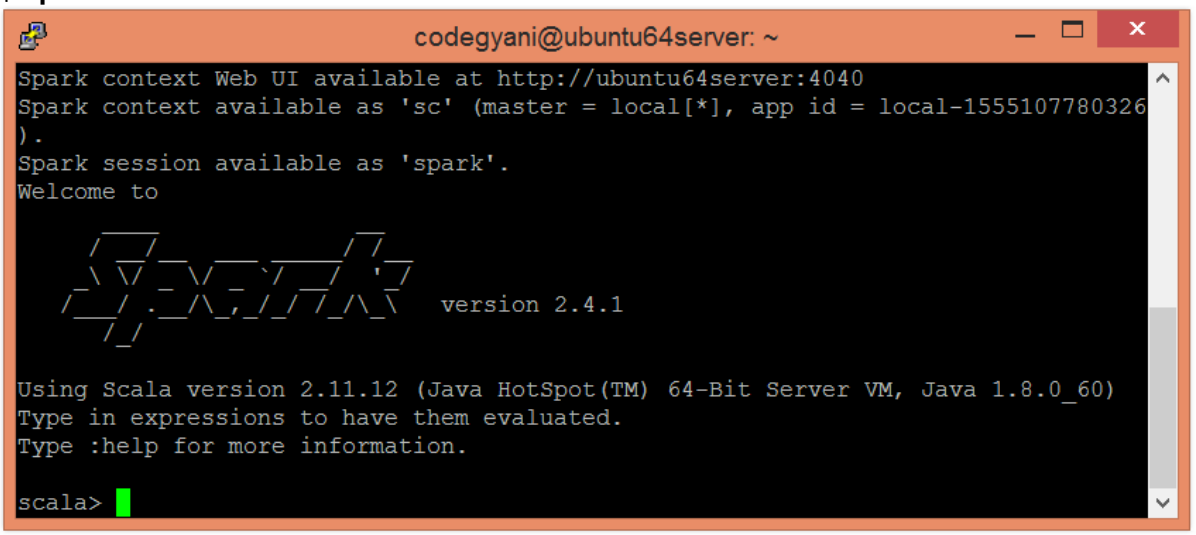
In Spark, the groupByKey function is a frequently used transformation operation that performs sl receives key-value pairs (K, V) as an input, group the values based on key and generates a dataset of (K an output.

Example of groupByKey Function

In this example, we group the values based on the key.

- **To open the Spark in Scala mode, follow the below command.**

1. **$ spark-shell**



   - **Create an RDD using the parallelized collection.**

1. **scala> val data = sc.parallelize(Seq(("C",3),("A",1),("B",4),("A",2),("B",5)))**

   Now, we can read the generated result by using the following command.

1. **scala> data.collect**

```
scala> val data = sc.parallelize(Seq(("C",3),("A",1),("B",4),("A",2),("B",5)))
data: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[6] at para
llelize at <console>:24

scala> data.collect
res5: Array[(String, Int)] = Array((C,3), (A,1), (B,4), (A,2), (B,5))

scala>
```

- **Apply groupByKey() function to group the values.**

1. **scala> val groupfunc = data.groupByKey()**
   - **Now, we can read the generated result by using the following command.**

1. **scala> groupfunc.collect**

```
scala> val groupfunc = data.groupByKey()
groupfunc: org.apache.spark.rdd.RDD[(String, Iterable[Int])] = ShuffledRDD[7] at
 groupByKey at <console>:25

scala> groupfunc.collect
res6: Array[(String, Iterable[Int])] = Array((B,CompactBuffer(4, 5)), (A,Compact
Buffer(1, 2)), (C,CompactBuffer(3)))

scala>
```

Here, we got the desired output.

Spark reduceByKey Function

In Spark, the reduceByKey function is a frequently used transformation operation that performs aggregation of data. It receives key-value pairs (K, V) as an input, aggregates the values based on the key and generates a dataset of (K, V) pairs as an output.

Example of reduceByKey Function

In this example, we aggregate the values on the basis of key.

- **To open the Spark in Scala mode, follow the below command.**

1. **$ spark-shell**

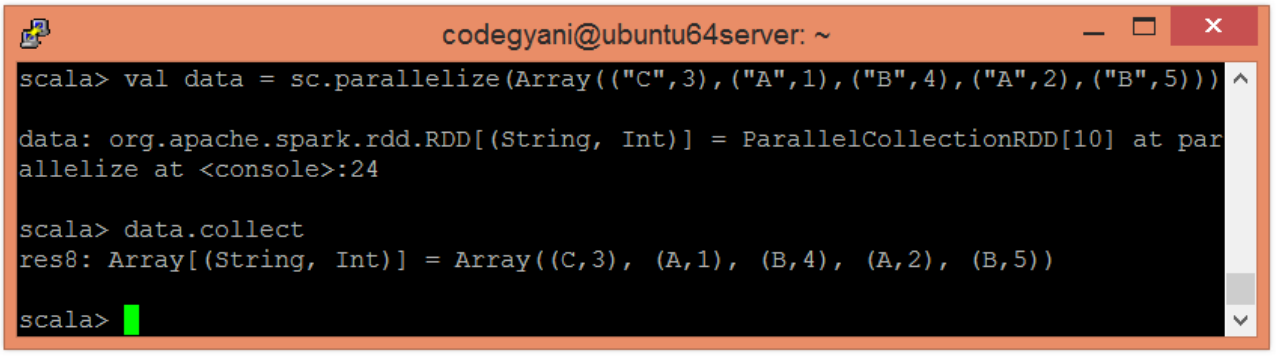
```
codegyani@ubuntu64server: ~
Spark context Web UI available at http://ubuntu64server:4040
Spark context available as 'sc' (master = local[*], app id = local-1555107780326
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_60)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

- **Create an RDD using the parallelized collection.**

1. **scala> val data = sc.parallelize(Array(("C",3),("A",1),("B",4),("A",2),("B",5)))**

   Now, we can read the generated result by using the following command.

1. **scala> data.collect**



   - **Apply reduceByKey() function to aggregate the values.**

1. **scala> val reducefunc = data.reduceByKey((value, x) => (value + x))**
   - **Now, we can read the generated result by using the following command.**

1. **scala> reducefunc.collect**

```
codegyani@ubuntu64server: ~

scala>  val reducefunc = data.reduceByKey((value, x) => (value + x))
reducefunc: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[11] at reduceB
yKey at <console>:27

scala> reducefunc.collect
res9: Array[(String, Int)] = Array((B,9), (A,3), (C,3))

scala>
```

Here, we got the desired output.

Spark cogroup Function

In Spark, the cogroup function performs on different datasets, let's say, (K, V) and (K, W) and return (Iterable, Iterable)) tuples. This operation is also known as groupWith.

Example of cogroup Function

In this example, we perform the groupWith operation.

- **To open the Spark in Scala mode, follow the below command.**

1. **$ spark-shell**

```
codegyani@ubuntu64server: ~
Spark context Web UI available at http://ubuntu64server:4040
Spark context available as 'sc' (master = local[*], app id = local-1555107780326
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_60)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

- **Create an RDD using the parallelized collection.**

1. **scala> val data1 = sc.parallelize(Seq(("A",1),("B",2),("C",3)))**

Now, we can read the generated result by using the following command.

1. **scala> data1.collect**

```
codegyani@ubuntu64server: ~

scala> val data1 = sc.parallelize(Seq(("A",1),("B",2),("C",3)))
data1: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[7] at par
allelize at <console>:24

scala> data1.collect
res4: Array[(String, Int)] = Array((A,1), (B,2), (C,3))

scala>
```

- **Create another RDD using the parallelized collection.**

1. **scala> val data2 = sc.parallelize(Seq(("B",4),("E",5)))**

Now, we can read the generated result by using the following command.

1. **scala> data2.collect**

```
codegyani@ubuntu64server: ~
scala> val data2 = sc.parallelize(Seq(("B",4),("E",5)))
data2: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[14] at pa
rallelize at <console>:24

scala> data2.collect
res8: Array[(String, Int)] = Array((B,4), (E,5))

scala>
```

- **Apply cogroup() function to group the values.**

1. **scala> val cogroupfunc = data1.cogroup(data2)**
   - **Now, we can read the generated result by using the following command.**

1. **scala> cogroupfunc.collect**

```
codegyani@ubuntu64server: ~

scala> cogroupfunc.collect
res9: Array[(String, (Iterable[Int], Iterable[Int]))] = Array((B,(CompactBuffer(
2),CompactBuffer(4))), (A,(CompactBuffer(1),CompactBuffer())), (C,(CompactBuffer
(3),CompactBuffer())), (E,(CompactBuffer(),CompactBuffer(5))))

scala>
```

Here, we got the desired output.

Spark First Function

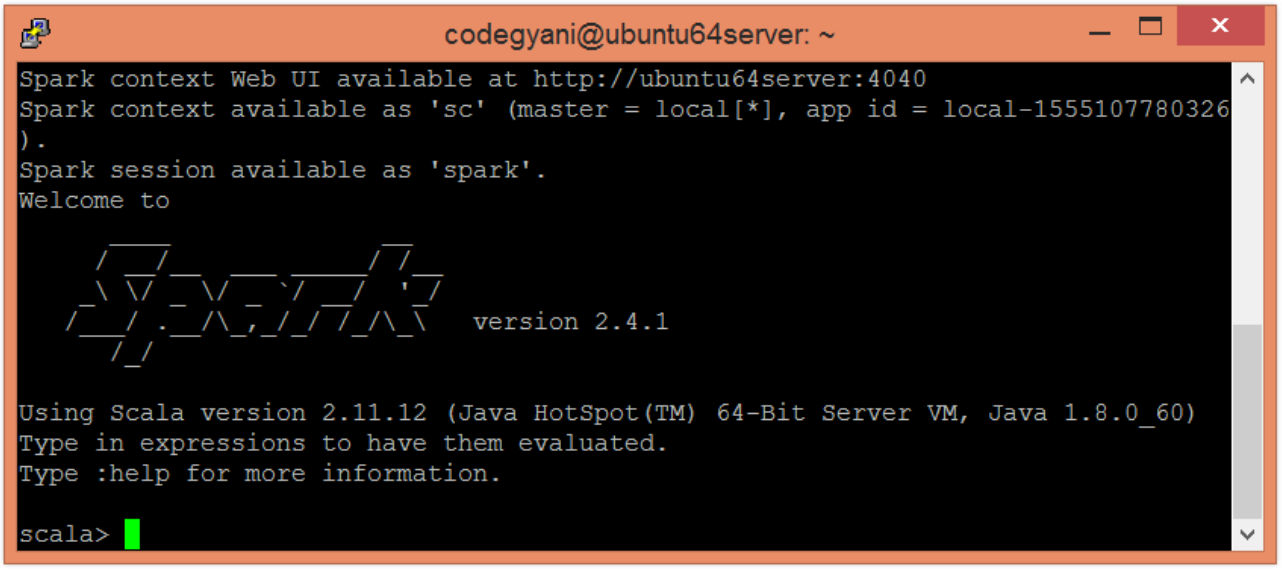
In Spark, the First function always returns the first element of the dataset. It is similar to take(1).

Example of First function

In this example, we retrieve the first element of the dataset.

- **To open the Spark in Scala mode, follow the below command.**

1. **$ spark-shell**



- **Create an RDD using the parallelized collection.**

1. **scala> val data = sc.parallelize(List(10,20,30,40,50))**
   - **Now, we can read the generated result by using the following command.**

1. **scala> data.collect**

```
codegyani@ubuntu64server: ~

scala> val data = sc.parallelize(List(10,20,30,40,50))
data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[1] at parallelize at
 <console>:24

scala> data.collect()
res1: Array[Int] = Array(10, 20, 30, 40, 50)

scala>
```

- **Apply first() function to retrieve the first element of the dataset.**

1. **scala> val firstfunc = data.first()**

```
codegyani@ubuntu64server: ~

scala> val firstfunc = data.first()
firstfunc: Int = 10

scala>
```

Here, we got the desired output.

Spark Take Function

In Spark, the take function behaves like an array. It receives an integer value (let say, n) as a parame array of first n elements of the dataset.

Example of Take function

In this example, we return the first n elements of an existing dataset.

- **To open the Spark in Scala mode, follow the below command.**

1. **$ spark-shell**

```
codegyani@ubuntu64server: ~
Spark context Web UI available at http://ubuntu64server:4040
Spark context available as 'sc' (master = local[*], app id = local-1555107780326
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.1
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_60)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

- **Create an RDD using the parallelized collection.**

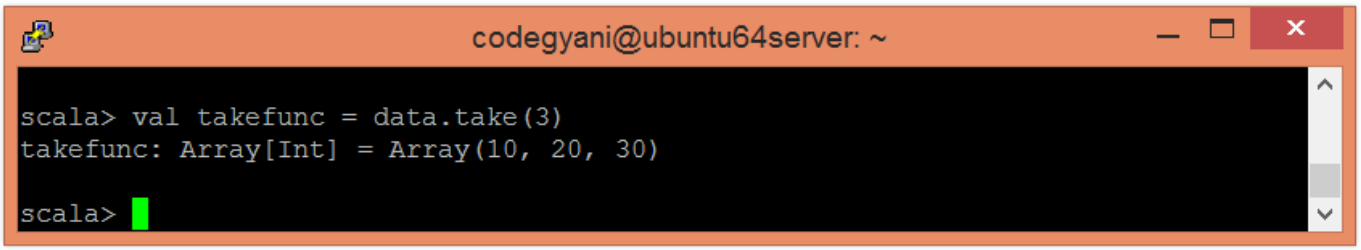
1. **scala> val data = sc.parallelize(List(10,20,30,40,50))**
   - **Now, we can read the generated result by using the following command.**

1. **scala> data.collect**

```
codegyani@ubuntu64server: ~

scala> val data = sc.parallelize(List(10,20,30,40,50))
data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[2] at paral
 <console>:24

scala> data.collect
res2: Array[Int] = Array(10, 20, 30, 40, 50)

scala>
```

- **Apply take() function to return an array of elements.**

1. **scala> val takefunc = data.take(3)**



Here, we got the desired output.

Next

Ne