**Aim: Install and Run Hive then use Hive to create, alter, and drop tables.**

hadoop\_user@Hadoop:~$ nano sample.txt

hadoop\_user@Hadoop:~$ cat sample.txt

1,abc

2,xyz

3,gbm

hadoop\_user@Hadoop:~$ hive

Hive Session ID = eadad695-5066-41ca-ae8b-c7c7f707a3dd

Logging initialized using configuration in

jar:file:/usr/local/hive/lib/hive-common-3.1.1.jar!/hive-log4j2.properties Async: true

Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider

using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.

Hive Session ID = 62e47107-fc20-4f8e-9e96-8a7b24b70233

hive> show databases;

OK

default

demo

Time taken: 0.491 seconds, Fetched: 2 row(s)

**a) Create table**

hive> create database demo;

FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask.

Database demo already exists

hive> create database demo1;

OK

Time taken: 0.095 seconds

hive> show databases;

OK

default

demo

demo1

Time taken: 0.04 seconds, Fetched: 3 row(s)

hive> describe database extended demo1;

OK

demo1 hdfs://localhost:9000/user/hive/warehouse/demo1.db hadoop\_user USER

Time taken: 0.089 seconds, Fetched: 1 row(s)

**b) Alter Table**

hive> create external table emplist (Id int, Name string)

> row format delimited

> fields terminated by ',';

FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask.

AlreadyExistsException(message:Table hive.default.emplist already exists)

hive> create external table emplist (Id int, Name string)

> row format delimited

> fields terminated by ','

> location '/HiveDir';

FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask.

AlreadyExistsException(message:Table hive.default.emplist already exists)

hive> create external table emplist1 (Id int, Name string)

> row format delimited

> fields terminated by ','

> location '/HiveDir';

OK

Time taken: 0.155 seconds

hive> load data local inpath 'sample.txt' overwrite into table emplist1;

Loading data to table default.emplist1

OK

Time taken: 1.037 seconds

hive> select \* from emplist1;

OK

1 abc

2 xyz

3 gbm

**Aim: Install and Run Pig then write Pig Latin scripts to sort, group, join, project, and filter the Weather data.**

**Load and dump**  
pig -x local

A = LOAD 'f1.txt' as(id:int,ornament:chararray);

dump A;

(,ornament)

(2,tie)

(4,coat)

(3,hat)

(1,scarf)

B = LOAD 'f2.txt' as(name:chararray,id:int);

dump B;

(name,)

(joe,2)

(hank,4)

(ali,0)

(eve,3)

(hank,2)

STORE A INTO 'd1' USING PigStorage(',');

describe A;

A: {id: int,ornament: chararray}

a = FOREACH A GENERATE $0,'constant';

dump a;

(,constant)

(2,constant)

(4,constant)

(3,constant)

(1,constant)

b = GROUP B BY SIZE($1);

dump b;

(1,{(hank,2),(eve,3),(ali,0),(hank,4),(joe,2)})

(,{(name,)})

b = GROUP B BY SIZE($0);

dump b;

(3,{(eve,3),(ali,0),(joe,2)})

(4,{(hank,2),(hank,4),(name,)})

**Join**

grunt> c = JOIN A BY $0,B BY $1;

dump c;

(2,tie,hank,2)

(2,tie,joe,2)

(3,hat,eve,3)

(4,coat,hank,4)

c = JOIN A BY $0 LEFT OUTER,B BY $1;

dump c;

(1,scarf,,)

(2,tie,hank,2)

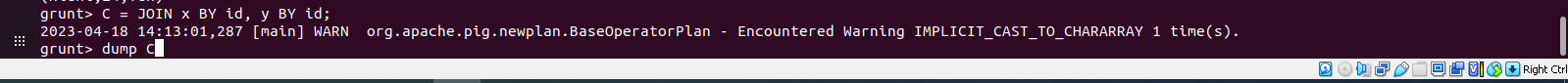
(2,tie,joe,2)

(3,hat,eve,3)

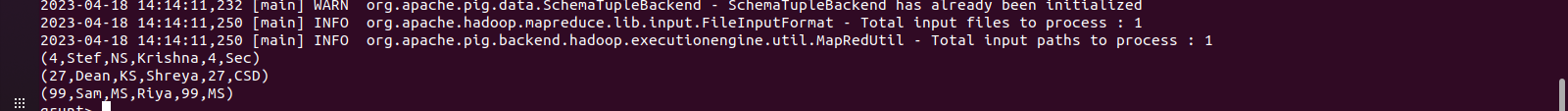
(4,coat,hank,4)

(,ornament,,)

**Join x and y by id**



output:



**GroupBy**

grunt> d = COGROUP A BY $0,B BY $1;

(0,{},{(ali,0)})

(1,{(1,scarf)},{})

(2,{(2,tie)},{(hank,2),(joe,2)})

(3,{(3,hat)},{(eve,3)})

(4,{(4,coat)},{(hank,4)})

(,{(,ornament)},{})

(,{},{(name,)})

d = COGROUP A BY $0 OUTER,B BY $1 OUTER;

(0,{},{(ali,0)})

(1,{(1,scarf)},{})

(2,{(2,tie)},{(hank,2),(joe,2)})

(3,{(3,hat)},{(eve,3)})

(4,{(4,coat)},{(hank,4)})

(,{(,ornament)},{})

(,{},{(name,)})

e = COGROUP A BY $0 INNER,B BY $1;

(1,{(1,scarf)},{})

(2,{(2,tie)},{(hank,2),(joe,2)})

(3,{(3,hat)},{(eve,3)})

(4,{(4,coat)},{(hank,4)})

(,{(,ornament)},{})

g = COGROUP A BY $0 INNER,B BY $1 INNER;

(2,{(2,tie)},{(hank,2),(joe,2)})

(3,{(3,hat)},{(eve,3)})

(4,{(4,coat)},{(hank,4)})

h = FOREACH g GENERATE FLATTEN($1),FLATTEN($2);

(2,tie,hank,2)

(2,tie,joe,2)

(3,hat,eve,3)

(4,coat,hank,4)

x= GROUP A ALL;

(all,{(1,scarf),(3,hat),(4,coat),(2,tie),(,ornament)})

y = UNION A,c;

(1,scarf,,)

(2,tie,hank,2)

(2,tie,joe,2)

(3,hat,eve,3)

(4,coat,hank,4)

(,ornament,,)

(,ornament)

(2,tie)

(4,coat)

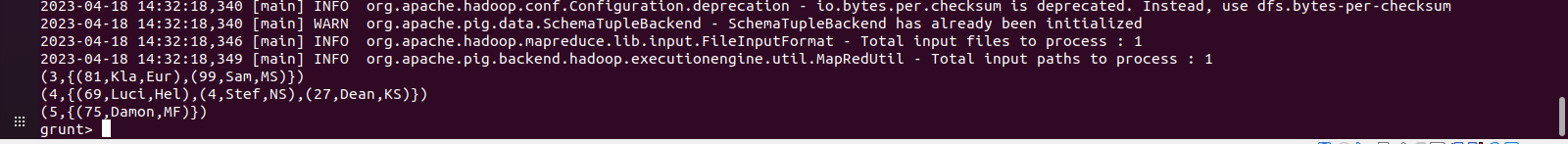
(3,hat)

(1,scarf)

**GroupBy (Size):**



Output:



**Order**

grunt> z = ORDER A BY $1,$0 desc;

(4,coat)

(3,hat)

(1,scarf)

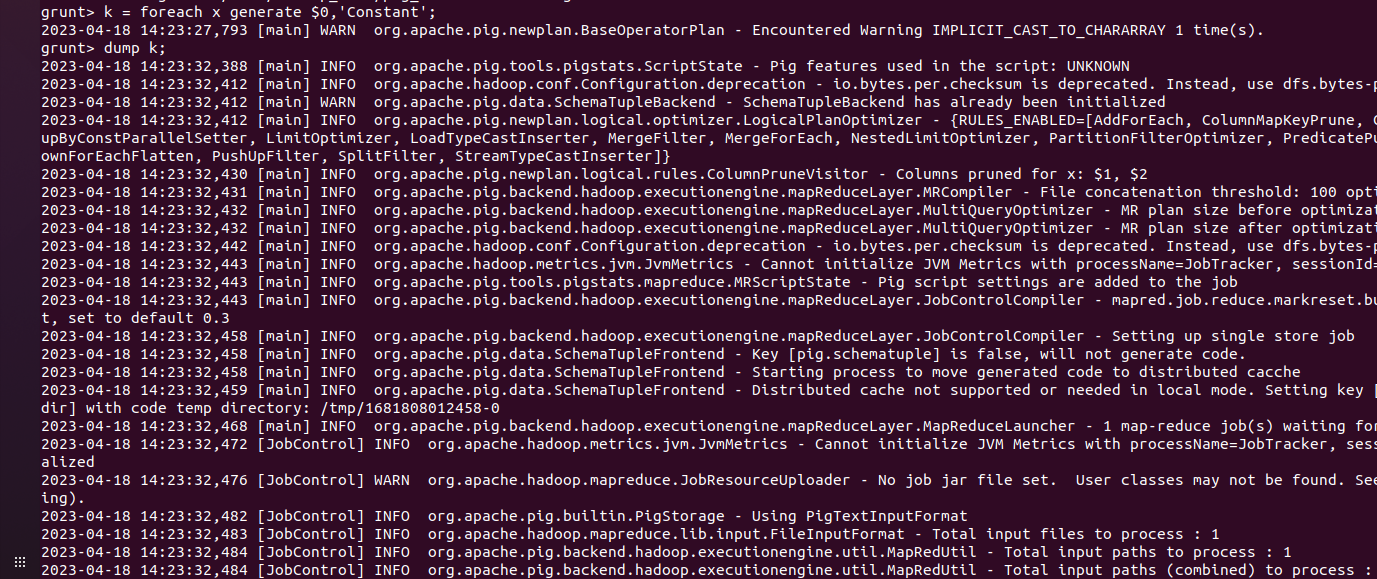
(2,tie)

w = LIMIT B 2;

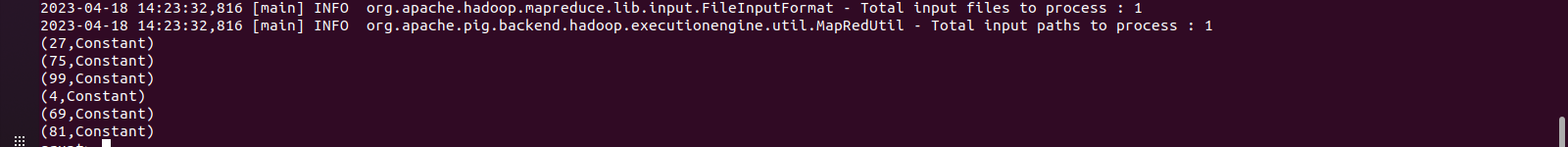
(name,)

(joe,2)

**foreach**

****

output:



**a) Aim: Install spark and write a Program for Developing and Running a Spark Word Count Application.**

**Commandline:**

scala> val data=sc.textFile("sparkdata.txt")

data: org.apache.spark.rdd.RDD[String] = sparkdata.txt MapPartitionsRDD[36] at textFile at

<console>:24

scala> data.collect

org.apache.hadoop.mapred.InvalidInputException: Input path does not exist:

file:/home/hadoop\_user/sparkdata.txt

at

org.apache.hadoop.mapred.FileInputFormat.singleThreadedListStatus(FileInputFormat.java:287

)

at org.apache.hadoop.mapred.FileInputFormat.listStatus(FileInputFormat.java:229)

at org.apache.hadoop.mapred.FileInputFormat.getSplits(FileInputFormat.java:315)

at org.apache.spark.rdd.HadoopRDD.getPartitions(HadoopRDD.scala:205)

at org.apache.spark.rdd.RDD.$anonfun$partitions$2(RDD.scala:276)

at scala.Option.getOrElse(Option.scala:189)

at org.apache.spark.rdd.RDD.partitions(RDD.scala:272)

at org.apache.spark.rdd.MapPartitionsRDD.getPartitions(MapPartitionsRDD.scala:49)

at org.apache.spark.rdd.RDD.$anonfun$partitions$2(RDD.scala:276)

at scala.Option.getOrElse(Option.scala:189)

at org.apache.spark.rdd.RDD.partitions(RDD.scala:272)

at org.apache.spark.SparkContext.runJob(SparkContext.scala:2179)

at org.apache.spark.rdd.RDD.$anonfun$collect$1(RDD.scala:1004)

at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)

at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112)

at org.apache.spark.rdd.RDD.withScope(RDD.scala:388)

at org.apache.spark.rdd.RDD.collect(RDD.scala:1003)

... 47 elided

scala> hadoop\_user@Hadoop:~$ gedit sparkdata.txt

hadoop\_user@Hadoop:~$ spark-shell

23/04/25 13:57:28 WARN Utils: Your hostname, Hadoop resolves to a loopback address:

…

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0\_291)

Type in expressions to have them evaluated.

Type :help for more information.

scala> val data=sc.textFile("sparkdata.txt")

data: org.apache.spark.rdd.RDD[String] = sparkdata.txt MapPartitionsRDD[1] at textFile at

<console>:24

scala> data.collect

[Stage 0:> (0 + 1)

res0: Array[String] = Array(hello to Hadoop, hello to Spark, hello to Hbase.)

scala> val splitdata = data.flatMap(line => line.split(" "));

splitdata: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at flatMap at <console>:25

scala> splitdata.collect

res1: Array[String] = Array(hello, to, Hadoop,, hello, to, Spark,, hello, to, Hbase.)

scala> val mapdata = splitdata.map(word => (word,1));

mapdata: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at

<console>:25

scala> mapdata.collect

res2: Array[(String, Int)] = Array((hello,1), (to,1), (Hadoop,,1), (hello,1), (to,1), (Spark,,1),

(hello,1), (to,1), (Hbase.,1))

**Reduce by key:**

scala> val reducedata = mapdata.reduceByKey((value,x)=>(value+x));

reducedata: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at

<console>:25

scala> reducedata.collect

res3: Array[(String, Int)] = Array((Hbase.,1), (Hadoop,,1), (to,3), (hello,3), (Spark,,1))

scala>

cala> val data=sc.textFile("sparkdata.txt")

data: org.apache.spark.rdd.RDD[String] = sparkdata.txt MapPartitionsRDD[7] at textFile at

<console>:24

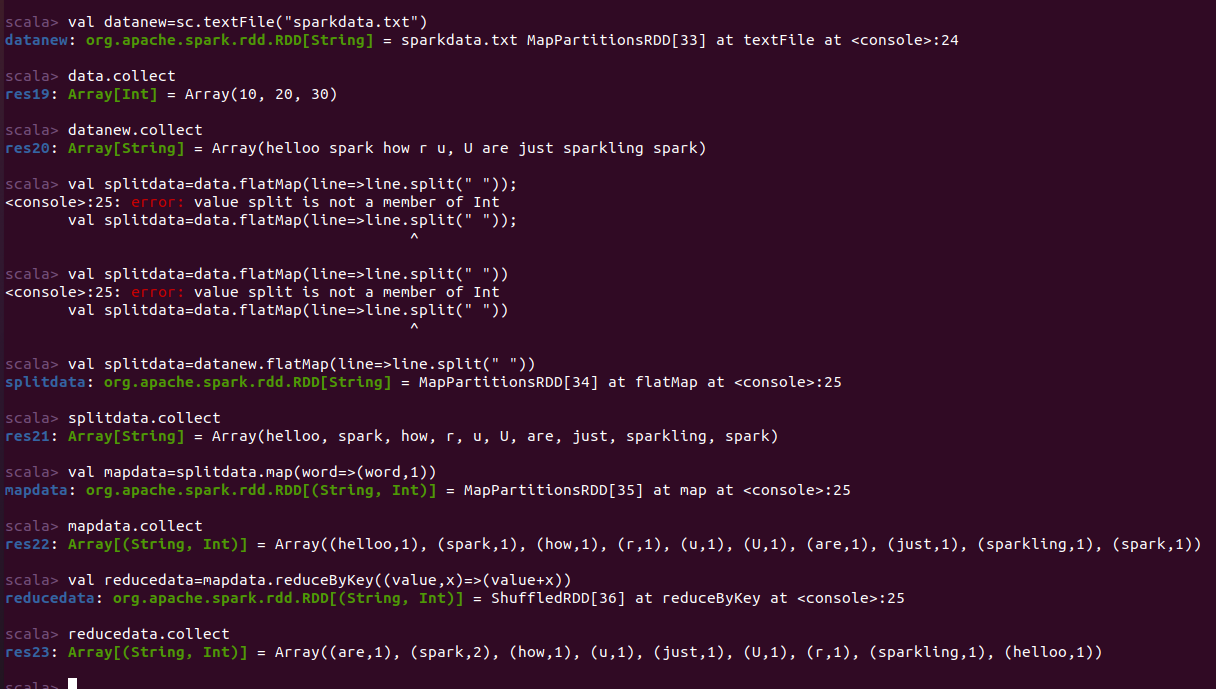
scala> data.collect

res5: Array[String] = Array(hello to Hadoop, hello to Spark, hello to Hbase.)

scala> data.saveAsTextFile("sparkfile.txt")

scala> data.collect

res7: Array[String] = Array(hello to Hadoop, hello to Spark, hello to Hbase.)



**Basic Spark commands**

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0\_291)

Type in expressions to have them evaluated.

Type :help for more information.

**Transformation and action function:**

scala> val data = sc.parallelize(List(10,20,30))

data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:24

scala> data.collect

res0: Array[Int] = Array(10, 20, 30)

**Map:**

scala> val mapfunc = data.map(x =&gt; x+10)

<console>:1: error: ')' expected but ';' found.

val mapfunc = data.map(x =&gt; x+10)

^

<console>:1: error: ';' expected but ')' found.

val mapfunc = data.map(x =&gt; x+10)

^

scala> val mapfunc = data.map(x => x+10)

mapfunc: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[1] at map at <console>:25

scala> mapfunc.collect

res1: Array[Int] = Array(20, 30, 40)

**Filter:**

scala> val data = sc.parallelize(List(10,20,30))

data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[2] at parallelize at <console>:24

scala> val filterfunc=data.filter(x=&>x!=30)

<console>:25: error: not found: value x

val filterfunc=data.filter(x=&>x!=30)

^

<console>:25: error: not found: value x

val filterfunc=data.filter(x=&>x!=30)

^

scala> val filterfunc=data.filter(x=>x!=30)

filterfunc: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[3] at filter at <console>:25

scala> filterfunc.collect

res2: Array[Int] = Array(10, 20)

**Count:**

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

data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[4] at parallelize at <console>:24

scala> val countfunc = data.count()

countfunc: Long = 5

**Distinct:**

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

data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[5] at parallelize at <console>:24

scala> val distinctfunc=data.distinct()

distinctfunc: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[8] at distinct at <console>:25

scala> distinctfunc.collect

res3: Array[Int] = Array(40, 20, 10)

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

<console>:24: error: value paralellize is not a member of org.apache.spark.SparkContext

val data1=sc.paralellize(List(1,2))

^

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

data1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[9] at parallelize at <console>:24

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

data2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at

<console>:24

scala> val unionfunc=data1.union(data2)

unionfunc: org.apache.spark.rdd.RDD[Int] = UnionRDD[11] at union at <console>:27

scala> unionfunc.collect

res4: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val data1=sc.parallelize(List(1,2,3))

data1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[12] at parallelize at

<console>:24

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

data2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[13] at parallelize at

<console>:24

**Intersect:**

scala> val intersectfunc=data1.intersection(data2)

intersectfunc: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[19] at intersection at

<console>:27

scala> intersectfunc.collect

res5: Array[Int] = Array(3)

**Cartesian:**

scala> val cartesianfunc=data1.cartesian(data2)

cartesianfunc: org.apache.spark.rdd.RDD[(Int, Int)] = CartesianRDD[20] at cartesian at

<console>:27

scala> cartesianfunc.collect

res6: Array[(Int, Int)] = Array((1,3), (1,4), (1,5), (2,3), (2,4), (2,5), (3,3), (3,4), (3,5))

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

<console>:1: error: unclosed string literal

val data = sc.parallelize(Seq(("C",3)("A",1)("D",4)("B",2)("

^

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

<console>:24: error: (String, Int) does not take parameters

Error occurred in an application involving default arguments.

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

^

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[21] at parallelize at

<console>:24

scala> data.collect

res7: Array[(String, Int)] = Array((C,3), (A,1), (D,4), (B,2), (E,5))

**Sort:**

scala> val sortfunc=data.sortByKey()

sortfunc: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[22] at sortByKey at

<console>:25

scala> data.sortByKey()

res8: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[23] at sortByKey at <console>:26

scala> sortfunc.collect

res9: Array[(String, Int)] = Array((A,1), (B,2), (C,3), (D,4), (E,5))

scala> val sortfunc=data.sortByKey(false)

sortfunc: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[24] at sortByKey at

<console>:25

scala> sortfunc.collect

res10: Array[(String, Int)] = Array((E,5), (D,4), (C,3), (B,2), (A,1))

**Group by:**

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[25] at parallelize at

<console>:24

scala> val groupfunc=data.groupByKey()

groupfunc: org.apache.spark.rdd.RDD[(String, Iterable[Int])] = ShuffledRDD[26] at groupByKey

at <console>:25

scala> groupfunc.collect

res11: Array[(String, Iterable[Int])] = Array((B,CompactBuffer(4, 5)), (A,CompactBuffer(1, 2)),

(C,CompactBuffer(3)))

**Reduce:**

scala> val reducefunc=data.reduceByKey((value,x)=>(value+x))

reducefunc: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[27] at reduceByKey at

<console>:25

scala> reducefunc.collect

res12: Array[(String, Int)] = Array((B,9), (A,3), (C,3))

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

data1: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[28] at parallelize at

<console>:24

scala> data1.collect

res13: Array[(String, Int)] = Array((A,1), (B,2), (C,3))

**CoGroup:**

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

<console>:1: error: ')' expected but string literal found.

val data2 = sc.parallelize(Seq(("B',4),("E",5)))

^

<console>:1: error: unclosed string literal

val data2 = sc.parallelize(Seq(("B',4),("E",5)))

^

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

data2: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[29] at parallelize at

<console>:24

scala> val cogroupfunc = data1.cogroup(data2)

cogroupfunc: org.apache.spark.rdd.RDD[(String, (Iterable[Int], Iterable[Int]))] =

MapPartitionsRDD[31] at cogroup at <console>:27

scala> cogroupfunc.collect

res14: Array[(String, (Iterable[Int], Iterable[Int]))] =

Array((B,(CompactBuffer(2),CompactBuffer(4))), (A,(CompactBuffer(1),CompactBuffer())),

(C,(CompactBuffer(3),CompactBuffer())), (E,(CompactBuffer(),CompactBuffer(5))))

**First:**

scala> val data = sc.parallelize(List(10,20,30,40,50))

data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[32] at parallelize at <console>:24

scala> val firstfunc = data.first()

firstfunc: Int = 10

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

data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[33] at parallelize at <console>:24

scala> data.collect

res15: Array[Int] = Array(1, 2, 3)

scala> val data1=data.reduce(lambda a, b: a \* b)

<console>:25: error: too many arguments (2) for method reduce: (f: (Int, Int) => Int)Int

val data1=data.reduce(lambda a, b: a \* b)

^

scala> val data1=data.reduce((a, b)=&gt; a \* b)

<console>:1: error: ')' expected but ';' found.

val data1=data.reduce((a, b)=&gt; a \* b)

^

<console>:1: error: ';' expected but ')' found.

val data1=data.reduce((a, b)=&gt; a \* b)

^

scala> val data1=data.reduce((a, b)=> a \* b)

data1: Int = 6

**Take:**

scala> val data = sc.parallelize(List(10,20,30,40,50))

data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[34] at parallelize at <console>:24

scala> val takefunc = data.take(3)

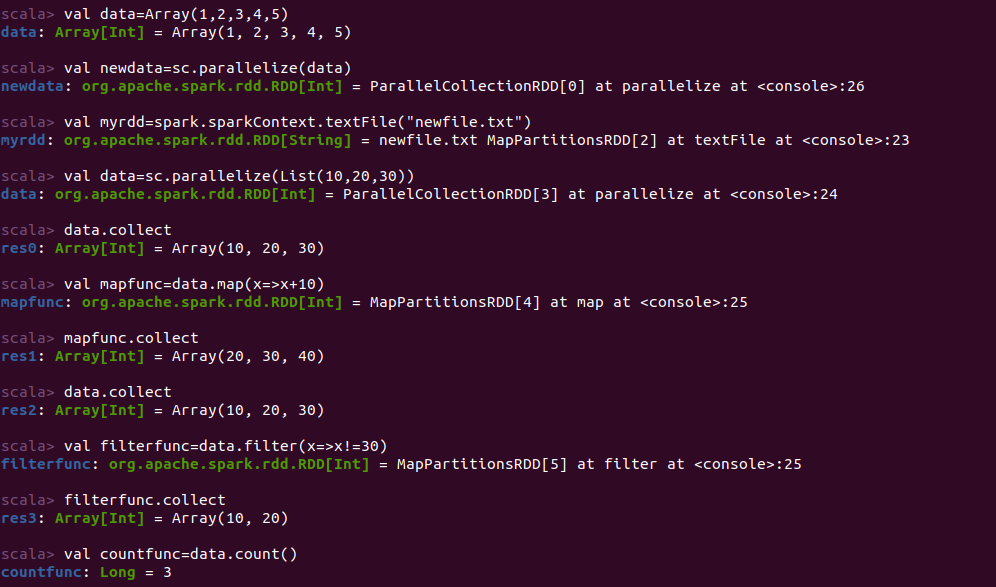
takefunc: Array[Int] = Array(10, 20, 30)

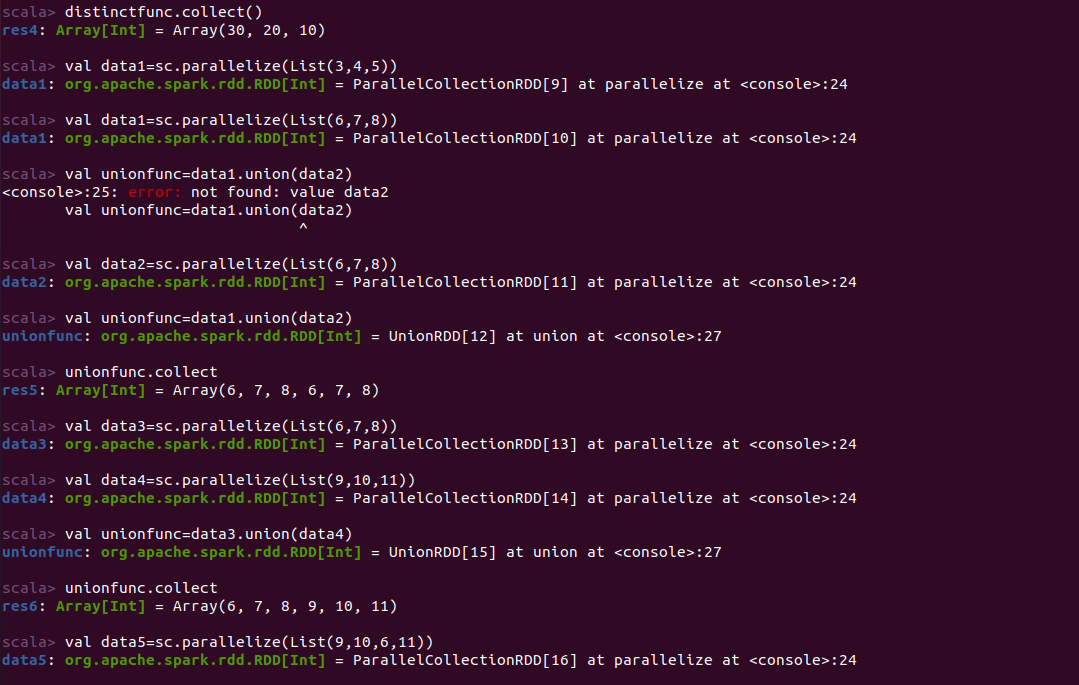
scala> val takefunc = data.take(1)

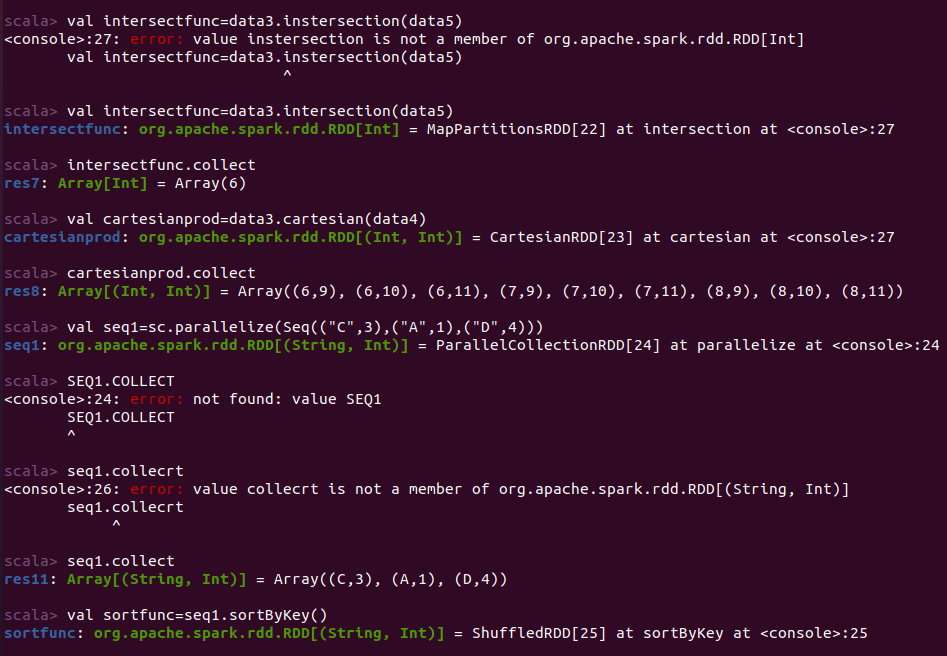
takefunc: Array[Int] = Array(10)

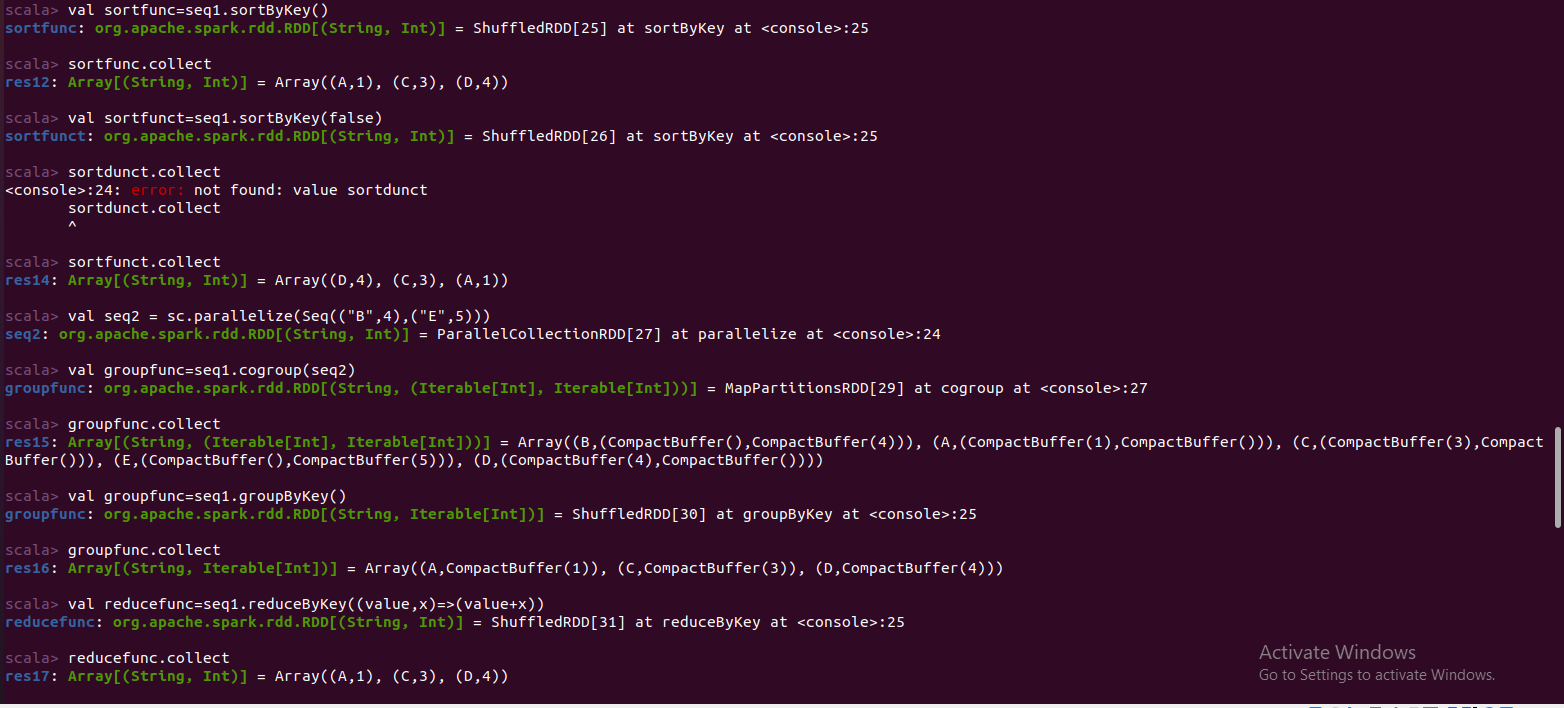
scala> val takefunc = data.take(5)

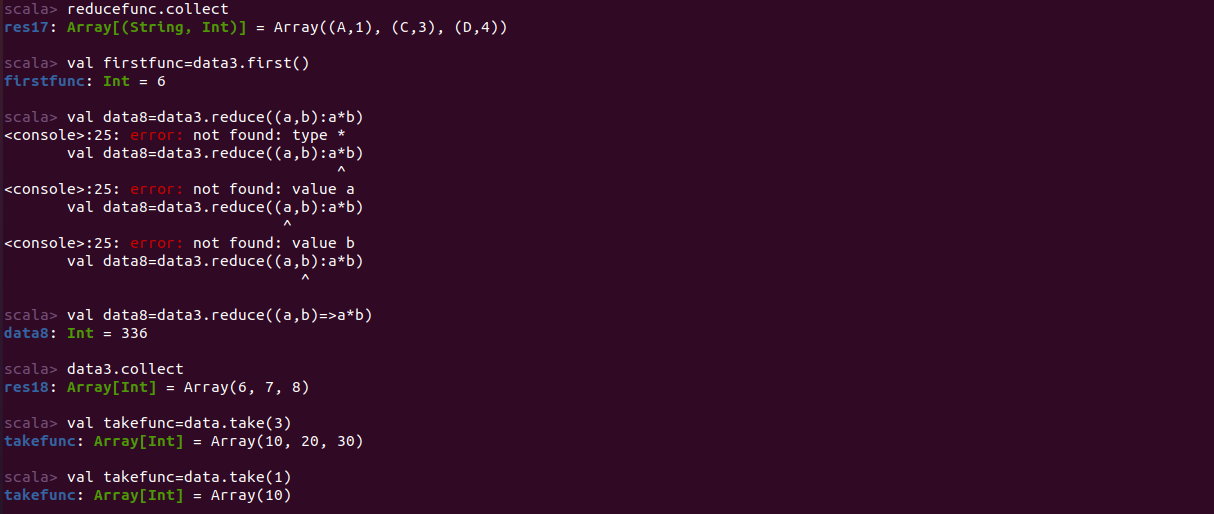
takefunc: Array[Int] = Array(10, 20, 30, 40, 50)











**b)Aim: Write a program for pattern matching using Scala.**

**Program:**

object Demo {

def main(args: Array[String]) {

println(“The number is :”+matchnumber(2))

println(“The symbol represents: “+matchsymbol(“?”))

}

def matchnumber(x: Int): String = x match {

case 1 => "one"

case 2 => "two"

case \_ => "many"

}

def matchsymbol(x:string):string=x match{

case “!” =>”Exclamation”;

case “?” =>”Question mark”

case \_ =>”other”

}

}

**CommandLine:**

\>scalac pattern.scala

\>scala pattern

**Output:**

The number is: two

The symbol represents: Question mark