**SPARK-** scala/python

**Broadcast Variables:**

|  |  |
| --- | --- |
| **val a =sc.broadcast(Array(1,2,3))**  **a.value** | **broadcastVar = sc.broadcast(list(range(1,4)))**  **broadcastVar.value** |

input.txt

-------------

And what is love? It is a doll dressed up

For idleness to cosset, nurse, and dandle;

A thing of soft misnomers, so divine

That silly youth doth think to make itself

Divine by loving, and so goes on

Yawning and doting a whole summer long,

Till Miss's comb is made a perfect tiara,

And common Wellingtons turn Romeo boots;

Till Cleopatra lives at Number Seven,

And Antony resides in Brunswick Square.

**Caching**

|  |  |
| --- | --- |
| **val a =sc.textFile("input.txt").map(x=>x.toUpperCase())**  **a.cache()**  **a.collect()**  **val b =a.filter(x=>x.startsWith("AND"))**  **b.collect()**  **b.count()** | **mydata = sc.textFile("input.txt")**  **myrdd = mydata.map(lambda s:s.upper())**  **myrdd.cache()**  **print(myrdd.collect())**  **myrdd2 = myrdd.filter(lambda s:s.startswith('AND'))**  **myrdd2.count()** |

**Map**

|  |  |
| --- | --- |
| **val x = sc.parallelize(Array("b","a","c"))**  **val y = x.map(x=>(x,1))**  **x.collect()**  **y.collect()** | **x = sc.parallelize(["b", "a", "c"])**  **y = x.map(lambda z: (z, 1))**  **print(x.collect())**  **print(y.collect())** |

**Filter**

|  |  |
| --- | --- |
| **val x = sc.parallelize(Array(1,2,3,4,5,6,7,8,9))**  **val y = x.filter(a=>a%2==1)**  **x.collect()**  **y.collect()** | **x = sc.parallelize([1,2,3,4,5,6,7,8,9,10])**  **y = x.filter(lambda x: x%2 == 1) #keep odd values**  **print(x.collect())**  **print(y.collect())** |

**JOIN**

|  |  |
| --- | --- |
| **val x = sc.parallelize(Array(("a", 1),("b", 2)))**  **val y = sc.parallelize(Array(("a", 3), ("a", 4), ("b", 5)))**  **val z = x.join(y)**  **z.collect()** | **x = sc.parallelize([("a", 1), ("b", 2)])**  **y = sc.parallelize([("a", 3), ("a", 4), ("b", 5)])**  **z = x.join(y)**  **print(z.collect())** |

coursedetails.txt

-------------------

bigdata hadoop informatica bigdata hadoop mainframe

sas bigdata hadoop mainframe

**Identify the Trending Technologies in the market (Sentimental Analytics)**

|  |  |
| --- | --- |
| **val counts = sc.textFile("coursedetails.txt").flatMap(line=>line.split(" ")).map(word=>(word,1)).reduceByKey((v1,v2)=>v1+v2)**  **counts.collect()** | **counts = sc.textFile('coursedetails.txt').flatMap(lambda line: line.split()).map(lambda word:(word,1)).reduceByKey(lambda v1,v2: v1+v2)**  **print(counts.collect())** |

**Spark SQL:**

sampledata.txt

1,a,1000

2,b,2000

3,c,3000

4,d,4000

|  |  |
| --- | --- |
| **val sqlContext = new org.apache.spark.sql.SQLContext(sc)**  **import sqlContext.implicits.\_** | **from pyspark.sql import SQLContext,Row**  **sqlCtx = SQLContext(sc)** |
| **val lines = sc.textFile("hdfs:///user/hduser/sampledata.txt")**  **val parts = lines.map(x=>x.split(","))** | **lines = sc.textFile("hdfs:///user/hduser/sampledata.txt")**  **parts = lines.map(lambda l:l.split(","))** |
| **case class Tab(id:Int,name:String,age:Int)**  **val people = parts.map(x=>Tab(x(0).toInt,x(1),x(2).toInt))** | **people = parts.map(lambda p: Row(id=p[0],name=p[1],sal=p[2]))** |
| **val peopleDF = people.toDF()** | **peopletable = sqlCtx.inferSchema(people) (or)**  **peopletable = sqlCtx.createDataFrame(people)** |
| **peopleDF.registerTempTable("peopleDF")** | **peopletable.registerAsTable("people")** |
| **peopleDF.show()**  **val peo = sqlContext.sql("SELECT name FROM peopleDF WHERE age>1000")**  **peo.collect()**  **peo.show()** | **teenagers = sqlCtx.sql("""SELECT name FROM people WHERE sal>=1000""")**  **print(teenagers.collect())**  **teenagers.show()** |

**Executing Script in python:**

**pyspark /home/hduser/wordcount.py**

**# Simple App to count word**

**#The following import statements imort SparkContext, SparkConf**

**from pyspark import SparkContext,SparkConf**

**Conf = SparkConf().setAppName("wordcount")**

**sc = SparkContext(conf=Conf)**

**counts = sc.textFile('coursedetails.txt').flatMap(lambda line: line.split()).map(lambda word:(word,1)).reduceByKey(lambda v1,v2: v1+v2)**

**print(counts.collect())**

**counts.saveAsTextFile("output")**