7.----------

```
import scala.io.Source
object scalaWordCount
{
  def main(args: Array[String])
   {
    if (args.length != 1)
    {
      System.err.println("Usage: sbt "run /Path/to/file.txt")
      System.exit(1)
    }
    val filename = args(0)
    val wordCount = scala.collection.mutable.Map[String, Int]()
    for (line <- Source.fromFile(filename).getLines)
      for (word <- line.split(" "))
            wordCount(word) = if (wordCount.contains(word)) wordCount(word) + 1 else 1
    for ( (k,v) <- wordCount)
      printf("Word %s occurs %d times\n", k,v)
    // println(wordCount)
  }
}
```

8.----------

```
import scala.io.StdIn
import scala.collection.mutable.ArrayBuffer
object MinMax {
  def main (args: Array[String] ): Unit = {
    var numArray = new ArrayBuffer[Int]()
    print("Enter number of elements: ")
    val n = scala.io.StdIn.readInt()//Read the number of items in Array
    println("Enter elements below, one per line.")
    for (i <- 0 until n)//Read the array elements
      numArray += scala.io.StdIn.readInt()
    // println(numArray)//Display the array elements
    val t = minmax(numArray)// Returned value will be tuple
    printf("Minimum number is %d\n", t._1)//Display the maximum
    printf("Maximum number is %d\n", t._2)//Display the minimum
  }
  def minmax(numArray:ArrayBuffer[Int]): (Int,Int) = {
    var mi:Int = numArray(0)//Initialize minumum and maximum
    var mx:Int = numArray(0)
    for ( i <- numArray)
    {
      if (i < mi)
            mi = i
      else if (i > mx)
```

```
            mx = i
        }
        (mi,mx)//Return mx and mn as items of tuple
    }
}
```

9.----------

```
object scalasortfuncstyle
{
  def sort(a: List[Int]): List[Int] = {
    if (a.length < 2) a
    else {
      val pivot = a(a.length / 2)
      sort(a.filter(_ < pivot)) :::
      a.filter(_ == pivot) :::
      sort(a.filter(_ > pivot))
    }
  }
  def main(args: Array[String])
 {
    val xs = List(6, 2, 8, 5, 1)
    println(xs)
    println(sort(xs))
 }
}
```

10.----------

// Matching on case classes: Case classes are especially useful for pattern matching.

// Define two case classes as below:

// abstract class Notification

// case class Email(sender: String, title: String, body: String) extends Notification

// case class SMS(caller: String, message: String) extends Notification

abstract class Notification

case class SMS(mobile:String, msg: String) extends Notification

case class Email(emailAddr: String, subject: String, body: String) extends Notification

// Define a function showNotification which takes as a parameter the abstract type Notification

// and matches on the type of Notification (i.e. it figures out whether it's an Email or SMS).

// In the case it's an Email(email, title, _) return the

// string: s"You got an email from $email with title: $title

// In the case it's an SMS return the String:

// s"You got an SMS from $number! Message: $message.

```
def showNotification(notification: Notification): String = {
  notification match {
    case Email(emailAddr, subject, _) =>
```

```
      s"You got an email from $emailAddr with subject: $subject"
    case SMS(number, message) =>
      s"You got an SMS from $number! Message: $message"
  }
}

// Test by creating an SMS instance and a Email instance.

val someSms = SMS("12345", "Are you there?")
val someEmail = Email("subhrajit.b@nmit.ac.in", "Big Data Course Syllabus",
  "Intro to Big Data, NOSQL Databases, Spark RDDs, SQL, Streaming")

println(showNotification(someSms))

println(showNotification(someEmail))
```

11. ----------

```
package com.nmit.spark.wordcount

import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD

/**
  * Problem statement:
  * Here the goal is to count how many times each word appears in a file and write out a list of
  * words whose count is strictly greater than 4.
  * Use the file log.txt accompanying this assignment to count the words.
 */

object wordcount {

  def main(args: Array[String]) {

    val pathToFile = "/home/subhrajit/sparkProjects/data/log.txt"

    // create spark configuration and spark context: the Spark context is the entry point in Spark.
    // It represents the connexion to Spark and it is the place where you can configure
    // the common properties
    // like the app name, the master url, memories allocation...
```

```
val conf = new SparkConf()
  .setAppName("Wordcount")
  .setMaster("local[*]")

val sc = new SparkContext(conf)

// load data and create an RDD where each element will be a word
// Here the flatMap method is used to separate the word in each line using the space separator
// You can experiment with "map" instead of "flatMap"to understand why flatMap is required.

val wordsRdd = sc.textFile(pathToFile)
  .flatMap(_.split(" "))

/**
  *  Now count how many times each word appears!
  */

// Step 1: the mapper step
// We want to attribute the number 1 to each word: so we create couples (word, 1).

val wordCountInitRdd = wordsRdd.map(word => (word, 1))

// Step 2: reducer step
```

```
    // Now you have a tuple (key, 1) where the key is a word,
    // you want to count the occurrences of (key, 1).
    // One way to do this is by using the reduce operation.

    val wordCountRdd = wordCountInitRdd.reduceByKey((v1, v2) => v1 + v2)
    // wordCountRdd.take(10).foreach(println)

    // Step 3: Filter Step
    // Now keep those words which appear strictly more than 4 times!
    // You can do this using the filter operation.

    val highFreqWords = wordCountRdd.filter(x => x._2 > 4)

    // save the word counts in a textfile "wordcountsDir".

    highFreqWords.saveAsTextFile("wordcountsDir")
  }
}
```

12.----------

```
package com.nmit.spark.tweetmining


import org.apache.spark.{SparkContext, SparkConf}


import org.apache.spark.rdd._


/**
 *
 *  Problem statement:
 *  We will use the dataset with the 8198 reduced tweets, reduced-tweets.json.
 *  The data are reduced tweets as the example below:
 *
 *  {"id":"572692378957430785",
 *  "user":"Srkian_nishu :)",
 *  "text":"@always_nidhi @YouTube no i dnt understand bt i loved of this mve is rocking",
 *  "place":"Orissa",
 *  "country":"India"}
 *
 *  We want to make some computations on the users:
 *  - find all the tweets by user
 *  - find how many tweets each user has
 *  - print the top 10 tweeters
 *
```

```
 */

object tweetmining {

  // Create the spark configuration and spark context
  val conf = new SparkConf()
    .setAppName("User mining")
    .setMaster("local[*]")

  val sc = new SparkContext(conf)

  // Needs an argument which is the path to the tweets file in json format.
  // e.g. "/home/subhrajit/sparkProjects/data/reduced-tweets.json"

  var pathToFile = ""

  def main(args: Array[String]) {
    if (args.length != 1) {
      println()
      println("Dude, I need exactly one argument.")
      println("But you have given me " + args.length +".")
      println("The argument should be path to json file containing a bunch of tweets. esired.")
      System.exit(1)
    }
```

```
pathToFile = args(0)

// The code below creates an RDD of tweets. Please look at the case class Tweet towards the end
// of this file.

val tweets =
  sc.textFile(pathToFile).mapPartitions(TweetUtils.parseFromJson(_))

// Create an RDD of (user, Tweet).
// Look at the tweet class. An object of type tweet will have fields "id", "user", "userName" etc.
// Collect all his tweets by the user. For this, you can use the field "user" of an object
// in the tweet RDD.
// Hint: the Spark API provides a groupBy method
// The code below should return RDD's with tuples (user, List of user tweets).

val tweetsByUser = tweets.map(x => (x.user, x)).groupByKey()
// tweets.take(10).foreach{ case(tweet) => println(tweet.user) }

// For each user, find the number of tweets he/she has made.
// Hint: we need tuples of (user, number of tweets by user)

val numTweetsByUser = tweetsByUser.map(x => (x._1, x._2.size))
```

```
    // Sort the users by the number of their tweets.

    val sortedUsersByNumTweets = numTweetsByUser.sortBy(_._2, ascending=false)

    // Find the Top 10 twitterers and print them to the console.

    sortedUsersByNumTweets.take(10).foreach(println)

  }
}

import com.google.gson._

object TweetUtils {
        case class Tweet (
                id : String,
                user : String,
                userName : String,
                text : String,
                place : String,
                country : String,
                lang : String
                )
```

```
        def parseFromJson(lines:Iterator[String]):Iterator[Tweet] = {
                val gson = new Gson
                lines.map(line => gson.fromJson(line, classOf[Tweet]))
        }
}

/**
  Whenever you have heavyweight initialization that should be done once for many RDD elements
  rather than once per RDD element, use mapPartitions() instead of map().
  mapPartitions() provides for the initialization to be done once per worker task/thread/partition
  instead of once per RDD data element for example.

  Example Scenario : if we have 100K elements in a particular RDD partition then we will fire
  off the function being used by the mapping transformation 100K times when we use map.

  Conversely, if we use mapPartitions then we will only call the particular function one time,
  but we will pass in all 100K records and get back all responses in one function call.

  There will be performance gain since map works on a particular function so many times,
  especially if the function is doing something expensive each time that it wouldn't need
  to do if we passed in all the elements at once(in case of mappartitions).
  * */
```

13.----------

```
package com.nmit.spark.ipltosswinstats

import org.apache.spark.sql.SparkSession

/**
  *  Problem Statement:
  *  We want to find the percentage of game wins by teams which win the toss.
  *  So lets say N games have been played.
  *  Let us say there are M games where the team which has won the toss has
  *  also won the game.
  *  It means that in (N-M) games, the team which won the toss lost the game.
  *  We are looking for the percentage (M * 100 / N).
  *  Perform the task using SQL code only.

  *  use the indian-premier-league-csv-dataset.
  * */

object ipltosswinstats {

  def main(args: Array[String]) {

    val pathToDB = "/home/subhrajit/sparkProjects/data/indian-premier-league-csv-dataset"
```

```
val sparkSession = SparkSession.builder().appName("My SQL Session").getOrCreate()
import sparkSession.implicits._

// The Match.csv file has the toss won/match won data for every game.
// Read the file into a dataframe.

val matchDF = sparkSession.read.format("csv").
  option("sep", ",").
  option("inferSchema", "true").
  option("header", "true").
  load(pathToDB + "/Match.csv")

// Since we have to use SQL queries, the dataframe has to be registered as a table.
// We can create a temporary table view.
matchDF.createOrReplaceTempView("matchStats")

// find the total number of entries in the table. this is equal to number of matches played or N.
val N = sparkSession.sql("SELECT COUNT(*) FROM matchstats")
  .first()(0)
  .asInstanceOf[Long]
// N.show()

// Find the subset of entries where the toss winner is also the match winner. This will be a dataframe.
```

```
    val tossNMatchwinnersDF = sparkSession.sql("SELECT * FROM matchstats WHERE Toss_Winner_Id = Match_Winner_Id")

    // register the dataframe as a temporary table so that you can use SQL queries on it.
    tossNMatchwinnersDF.createOrReplaceTempView("tossNMatchwinners")

    // find the count of entries in this Table. This gives us M.
    val M = sparkSession.sql("SELECT COUNT(*) FROM tossNMatchwinners")
      .first()(0)
      .asInstanceOf[Long]
    // M.show()

    // print M * 100 / N.
    println("Percentage of times Toss Winners have won the match = " + (M*100.0)/N + "%")
  }
}
```

14.----------

```
package com.nmit.spark.sparkwindowedstreaming


import org.apache.spark.sql.functions._
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.{window, col}
import org.apache.spark.sql.types.StructType


object sparkWindowedStreaming {
  def main(args: Array[String]) {


    val spark = SparkSession.builder.appName("sparkWindowedStreaming").getOrCreate()


    import spark.implicits._


    spark.conf.set("spark.sql.shuffle.partitions", 5)


    val userSchema = new StructType().
      add("Creation_Time", "double").
      add("Station", "string").
      add("Rainfall", "float")


    val streaming = spark.
      readStream.
```

```
    schema(userSchema).
    json("/home/subhrajit/sparkProjects/data/event-data/threeWindows")

  val withEventTime = streaming.selectExpr(
    "*",
    "cast(cast(Creation_Time as double)/1000000000 as timestamp) as event_time")

  val events_per_window = withEventTime
    .groupBy(window(col("event_time"), "15 minutes"))
    .agg(avg("Rainfall"),count("Station"))
    .writeStream
    .queryName("events_per_window")
    .format("console")
    .outputMode("complete")
    .option("truncate", false)
    .start()

  events_per_window.awaitTermination()

  }
}

// .withWatermark("event_time", "1 minutes")
// .groupBy(window(col("event_time"), "15 minutes", "5 minutes"))
```

```
// val events_per_window = withEventTime
    // .agg(avg("Rainfall"))
    // .writeStream
    // .queryName("events_per_window")
    // .format("console")
    // .outputMode("complete")
    // .option("truncate", false)
    // .start()
```