EJERCICIO 1.

Crea un nuevo RDD con 3 particiones a partir del siguiente array de números. Calcula:

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

object ejemplo {
  def main( args: Array[String]):Unit={
    val conf = new SparkConf().setAppName("primera").setMaster("local[*]")

    val sc = new SparkContext(conf)

    val array = Array(23,54,32,66,0,-14,-8,10)
    val numerosRDD = sc.parallelize(array,3)


    //println("El valor máximo dentro de numerosRDD es " + numerosRDD.max() + " y el mínimo
" + numerosRDD.min())
    println("La suma de todos los elementos es: " + numerosRDD.reduce((x,y) => x+y))
  }

}
```

El valor máximo dentro de numerosRDD es 66 y el mínimo -14

La suma de todos los elementos es: 163

EJERCICIO 2

Realiza un filtrado del RDD anterior quedándonos únicamente con las palabras de tamaño 6 o superior. Luego, transforma esas palabras a mayúsculas

```
import org.apache.spark.SparkConf

import org.apache.spark.SparkContext

import org.apache.spark.rdd.RDD



object ejemplo {

  def main( args: Array[String]):Unit={

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

```
   val sc = new SparkContext(conf)


   //carga del fichero loremipsum.txt

   val lorem = sc.textFile("lectura.txt")


val mayor6 = lorem.flatMap(line => line.split(" ")).filter(p => p.length() >= 6)

   val maymayor6 = mayor6.map(x => x.toUpperCase())

   val lista=maymayor6.collect().foreach(println)

 }
```

Ejercicio3 Dada una lista de animales ("Perro","Gato","Tigre","Caballo","Camello","Perro") debemos mostrar de todos ellos los que no sean de una categoría concretar (en este caso queremos eliminar de la lista al “Perro”)

```
import org.apache.spark.SparkConf

import org.apache.spark.SparkContext

import org.apache.spark.rdd.RDD


object ejemplo {

  def main( args: Array[String]):Unit={

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

   val sc= new SparkContext(conf)

   val listaAnimales = List("Perro","Gato","Tigre","Caballo","Camello","Perro")

   //todas las transformaciones de un RDD son lazy

   val animalesRDD = sc.parallelize(listaAnimales)

   val animalesSinRepeticionesSinPerros = animalesRDD.distinct().filter(animal => animal !=
"Perro").collect()

   for(animal <- animalesSinRepeticionesSinPerros) {

     println(animal)

   }

   sc.stop()  }}
```

EJERCICIO3

EJERCICIO1. Se nos facilita una lista formada por pares K-V como la siguiente: Lista=(("11111111H",39.1),("22222222H",34.2),("11111111H",39.6),("33333333H",37.4),("22222222H",36.5)) .

Queremos que nuestro código nos muestre por cada cliente de mi lista la cantidad de dinero que se gastado en las compras de navidad.

Debemos resolverlo usando GROUPBYKEY y después REDUCEBYKEY

```
import org.apache.spark.SparkConf

import org.apache.spark.SparkContext

import org.apache.spark.rdd.RDD

object ejemplo {

  def main( args: Array[String]):Unit={

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

    val sc= new SparkContext(conf)

    val cliente =
List(("11111111H",39.1),("22222222H",34.2),("11111111H",39.6),("33333333H",37.4),("22222222H",36.5))

    val clienteRDD = sc.parallelize(cliente) //RDD

    //GROUPBYKEY

    /*

    ("11111111H",[39.1,39.6]) -> ("11111111H",(39.1+39.6))

    ("22222222H",[34.2,36.5])

    ("33333333H",[37.4])

    * */

    val valores = clienteRDD.groupByKey()

      .map(cliente=> (cliente._1,cliente._2.sum))

    valores.foreach(println)
```

```
  //REDUCEKEY

  val valoresReduce = clienteRDD.reduceByKey((a,b) => a + b).collect()

  valoresReduce.foreach(println)

 sc.stop()   } }
```

EJERCICIO 5

Haciendo uso del fichero “LorenIpsum.txt”, carga su contenido en un RDD. Luego, implementa **un cuenta palabras**. Mostrar por cada clave su valor

```
//importamos SparkConf, SparkContext, RDD

import org.apache.spark.SparkConf

import org.apache.spark.SparkContext

import org.apache.spark.rdd.RDD



object ejemplo {

 def main( args: Array[String]):Unit={

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

  val sc = new SparkContext(conf)

  //carga del fichero loremipsum.txt

  val lorem = sc.textFile("lectura.txt")

  //dividir por palabras

  val palabras = lorem.flatMap(line => line.split(" "))

    //map para contar crear el par clave-valor para cada palabra

    .map(word => (word,1))

    //numero de ocurrencias por cada clave

    .reduceByKey(_+_)

 print(palabras.collectAsMap())



}
```

EJERCICIO 6 LEER EL COTENIDO DE UN FICHERO

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext

import org.apache.spark.sql.Dataset
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.SparkSession

object ejemplo {
  def main(args: Array[String]): Unit = {
al spark:SparkSession = SparkSession.builder()
     .master("local[*]")
     .appName("lectura TXT ")
     .getOrCreate()

   spark.sparkContext.setLogLevel("ERROR")    val rdd3 =
spark.sparkContext.textFile("lectura.txt")
   rdd3.foreach(println)
```

EJERCICIO7 LEER LOS FICHEROS QUE CUMPLAN ESE PATRON CON * ¿ ETC…

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext

import org.apache.spark.sql.Dataset
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.SparkSession

object ejemplo {
```

```
  def main(args: Array[String]): Unit = {
al spark:SparkSession = SparkSession.builder()
      .master("local[*]")
      .appName("lectura  TXT CON PATRON ")
      .getOrCreate()

    spark.sparkContext.setLogLevel("ERROR")
val rdd3 = spark.sparkContext.textFile("lec*.txt")
    rdd3.foreach(println)
```