

**Aims:**

This exercise aims to get you to:

1. Practice with the use of Apache Spark and its shell
2. Implement and run the example WordCount using Apache Spark
3. Practice with common Transformations and Actions in Spark.

## Run Apache Spark on VLab

Apache Spark has been installed on the lab computer. In order to do this, first you need to open a terminal in VLab and run the following program:

\$ 9313

This program will setup the environment needed to run your program on Hadoop. You should see an output like the following:

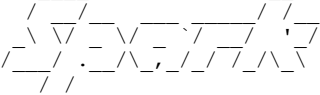
```
$ 9313
Welcome to COMP9313!
newclass starting new subshell for class COMP9313...
```

Next, run the spark-shell, which is an interactive tool that allows you to write Spark programs using Scala language:

```
$ spark-shell
```

...

Welcome to



version 2.4.3

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 11.0.3)  
Type in expressions to have them evaluated.  
Type :help for more information.

```
scala>
```

## Create and manipulate an RDD starting from a Scala array

As a warm-up, let's create an RDD from an Scala array:

```
scala> val myList = sc.parallelize(Array(1,2,3,4,5))

myList:  org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[3] at parallelize at
<console>:24
```

Now, let's create a new list derived from `myList` above, where the new list contains the same elements divided by 2.0.

```
scala> val myNewList = myList.map(x => x/2.0)

myNewList: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[4] at map at <console>:25
```

We can now print `myNewList` on the screen to check the results (order of output might be different in your shell):

```
scala> myNewList.foreach(println)
0.5
1.0
1.5
2.0
2.5
```

Let's print the original list from which `myList` was derived (order of output might be different in your shell):

```
scala> myList.foreach(println)
1
2
3
4
5
```

Notice that the original list `myList` was not modified after the map transformation was applied to it.

## Word Count in Spark

Let's now implement the word count example in Spark. We assume below that you have a text file named `myInputFile.txt` in the file system (you can create a text file yourself for testing purposes). Notice that you might need to provide the full path of your file (as stored in the file system)

```
scala> val file = sc.textFile("myInputFile.txt", 1)

file: org.apache.spark.rdd.RDD[String] = myInputFile.txt MapPartitionsRDD[1] at textFile at <console>:24

scala> val words = file.flatMap(line => line.split(" "))

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

scala> val pairs = words.map(word => (word, 1))

pairs: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at <console>:25

scala> val counts = pairs.reduceByKey(_ + _)

counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <console>:25

scala> counts.saveAsTextFile("myOutput")
```

After executing the command above, you should end up with a directory named `myOutput` in your file system. Inside this directory, you will find the file(s) containing the final result of the computations above.

## Sample of Spark Transformations

***map(func):*** returns a new RDD formed by passing each element of the source through a function ***func***.

```
scala> val myList = sc.parallelize(Array(1,2,3,4,5,6,7,8,9,10))

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

scala> val myListInc2 = myList.map(x => x + 2)

myListInc2: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[1] at map at <console>:25

scala> myListInc2.foreach(println)
3
4
5
6
7
8
9
10
11
12
```

***filter(func):*** returns a new RDD formed by selecting those elements of the source on which ***func*** returns true.

```
scala> val divByTwo = myList.filter(x => x % 2 == 0)

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

scala> divByTwo.foreach(println)
2
4
6
8
10
```

***distinct():*** retruns new RDD that contains the distinct elements of the source dataset

```
scala> val repList = sc.parallelize(Array("hello", "world", "hello", "unsw"))

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

scala> val uniqList = repList.distinct()

uniqList: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[12] at distinct at <console>:25

scala> uniqList.foreach(println)
hello
world
unsw
```

## Sample of Spark Actions

***reduce(func):*** aggregates RDD's elements using function **func**.

```
scala> val salaries = sc.parallelize(Array(10000, 12000, 3000, 15500))

salaries: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[39] at parallelize at
<console>:24

scala> val totalSalary = salaries.reduce(_ + _)

totalSalary: Int = 40500
```

***take(n):*** returns an array with the first **n** elements.

```
scala> val myList = sc.parallelize(1 to 1000)

myList: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[42] at parallelize at
<console>:24

scala> myList.take(100)

res31: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64,
65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100)
```

***collect():*** returns all the elements as an array

```
scala> val myList = sc.parallelize(1 to 50)

myList: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[43] at parallelize at
<console>:24

scala> myList.collect()

res32: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44, 45, 46, 47, 48, 49, 50)
```

***takeOrdered(n, key=func):*** returns **n** elements ordered in ascending order or as specified by the optional **key** function.

```
scala> val myList = sc.parallelize(Array(5,7,2,8,9,1,3,22,13))

myList: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[44] at parallelize at
<console>:24

scala> myList.takeOrdered(5)

res33: Array[Int] = Array(1, 2, 3, 5, 7)
```