SCALA SAMPLE

CARTESIAN() :-

```
scala> val input1 = sc.parallelize(List(1,2,3,4))
input1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[54] at paralleliz
e at <console>:24

scala> val input2 = sc.parallelize(List(3,4,5))
input2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[55] at paralleliz
e at <console>:24

scala> val cartesianOutput = input1.cartesian(input2)
cartesianOutput: org.apache.spark.rdd.RDD[(Int, Int)] = CartesianRDD[56] at car
tesian at <console>:27

scala> println(cartesianOutput.collect().mkString(","))
(1,3),(1,4),(1,5),(2,3),(2,4),(2,5),(3,3),(3,4),(3,5),(4,3),(4,4),(4,5)

scala> val cartesianOutput = input2.cartesian(input1)
cartesianOutput: org.apache.spark.rdd.RDD[(Int, Int)] = CartesianRDD[57] at car
tesian at <console>:27

scala> println(cartesianOutput.collect().mkString(","))
(3,1),(3,2),(3,3),(3,4),(4,1),(4,2),(4,3),(4,4),(5,1),(5,2),(5,3),(5,4)

scala>
```

DISTINCT() :-

```
scala> val input = sc.parallelize(List(1,2,3,4,2,4))
input: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[33] at parallelize
 at <console>:24

scala> val distinctOutput = input.distinct()
distinctOutput: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[36] at distinc
t at <console>:25

scala> println(distinctOutput.collect().mkString(","))
4,1,3,2
```

FLATMAP() :-

```
scala> val input = sc.parallelize(List("hello world","hi"))
input: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[4] at paralleli
ze at <console>:24

scala> val result = input.flatMap(x=>x.split(" "))
result: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[5] at flatMap at <c
onsole>:25

scala> result.first()
res1: String = hello
```

INTERSECTION() :-

```
scala> val input1 = sc.parallelize(List(1,2,3,4))
input1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[23] at paralleliz
e at <console>:24

scala> val input2 = sc.parallelize(List(3,4,5,6,7))
input2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[24] at paralleliz
e at <console>:24

scala> val intersectOutput = input1.intersection(input2)
intersectOutput: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[30] at inters
ection at <console>:27

scala> println(intersectOutput.collect().mkString(","))
4,3
```

MAP() :-

```
scala> val input = sc.parallelize(List(1,2,3,4))
input: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[2] at parallelize
at <console>:24

scala> val result = input.map(x=>x*x)
result: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[3] at map at <console>
:25

scala> println(result.collect().mkString(","))
[Stage 0:>                                                          (0 + 0) / 1
[Stage 0:>                                                          (0 + 1) / 1

1,4,9,16
```

SUBTRACT() :-

```
scala> val input1 = sc.parallelize(List(1,2,3,4))
input1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[44] at paralleliz
e at <console>:24

scala> val input2 = sc.parallelize(List(3,4,5,6,7))
input2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[45] at paralleliz
e at <console>:24

scala> val subOutput = input1.subtract(input2)
subOutput: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[49] at subtract at
<console>:27

scala> println(subOutput.collect().mkString(","))
1,2

scala> val subOutput = input2.subtract(input1)
subOutput: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[53] at subtract at
<console>:27

scala> println(subOutput.collect().mkString(","))
5,6,7
```

UNION() :-

```
scala> val input1 = sc.parallelize(List(1,2,3,4))
input1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[18] at paralleliz
e at <console>:24

scala> val input2 = sc.parallelize(List(5,6,7))
input2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[19] at paralleliz
e at <console>:24

scala> val unionOutput = input1.union(input2)
unionOutput: org.apache.spark.rdd.RDD[Int] = UnionRDD[20] at union at <console>
:27

scala> println(unionOutput.collect().mkString(","))
1,2,3,4,5,6,7
```