reduce

Obtener el máximo del RDD

```
def miFuncion(acumulado, valor):
      max = acumulado
      if valor > max:
            max = valor
      return max

miRDD = sc.parallelize([4, 2, 7,
                        15, 4, 2,
                        3, 16, 3],3)

salidaDriver = miRDD.reduce(miFuncion)

print("\nmiRDD:", miRDD.collect(),"\nSalida en driver:", salidaDriver,
"\n")
```

collect

Transformamos el RDD en un array

```
miRDD = sc.parallelize([4, 2, 7,
                        15, 4, 2,
                        3, 16, 3],3)

salidaDriver = miRDD.collect()

print("\nmiRDD:", miRDD.collect(),"\nSalida en driver:", salidaDriver,
"\n")
```

count

Contamos la cantidad de elementos del RDD

```
miRDD = sc.parallelize([4, 2, 7,
                        15, 4, 2,
                        3, 16, 3],3)

salidaDriver = miRDD.count()

print("\nmiRDD:", miRDD.collect(),"\nSalida en driver:", salidaDriver,
"\n")
```

first

Obtenemos el primer elemento

```
miRDD = sc.parallelize([4, 2, 7,
                        15, 4, 2,
```

```
                        3, 16, 3],3)

salidaDriver = miRDD.first()

print("\nmiRDD:", miRDD.collect(),"\nSalida en driver:", salidaDriver,
"\n")
```

take

Obtener los 5 primeros elementos

```
miRDD = sc.parallelize([4, 2, 7,
                        15, 4, 2,
                        3, 16, 3],3)

salidaDriver = miRDD.take(5)

print("\nmiRDD:", miRDD.collect(),"\nSalida en driver:", salidaDriver,
"\n")
```

takeSample

Obtener aleatoriamente 5 elementos

takeSample sin reemplazamiento

```
miRDD = sc.parallelize([4, 2, 7,
                        15, 4, 2,
                        3, 16, 3],3)

salidaDriver = miRDD.takeSample(False, 5)

print("\nmiRDD:", miRDD.collect(),"\nSalida en driver:", salidaDriver,
"\n")
```

takeSample con reemplazamiento

```
miRDD = sc.parallelize([4, 2, 7,
                        15, 4, 2,
                        3, 16, 3],3)

salidaDriver = miRDD.takeSample(True, 5)

print("\nmiRDD:", miRDD.collect(),"\nSalida en driver:", salidaDriver,
"\n")
```

takeSample con seed

```
miRDD = sc.parallelize([4, 2, 7,
                        15, 4, 2,
                        3, 16, 3],3)

salidaDriver = miRDD.takeSample(False, 5, 1)

print("\nmiRDD:", miRDD.collect(),"\nSalida en driver:", salidaDriver,
"\n")
```

takeOrdered

Obtener los primeros 5 números según el orden natural

```
miRDD = sc.parallelize([4, 2, 7,
                        15, 4, 2,
                        3, 16, 3],3)

salidaDriver = miRDD.takeOrdered(5)

print("\nmiRDD:", miRDD.collect(),"\nSalida en driver:", salidaDriver,
"\n")
```

saveAsTextFile

Guarda un RDD

saveAsTextFile en el dfs por defecto

```
miRDD = sc.parallelize([4, 2, 7,
                        15, 4, 2,
                        3, 16, 3],3)

miRDD.saveAsTextFile("datasetGuardado")
```

Nota: si no tenemos un DFS asignado por defecto, nos lo guardará en una carpeta local. Si tenemos por ejemplo HDFS, entonces nos lo guardará en HDFS. También podríamos utilizar hdfs:///user/hadmin/datasetGuardado

Si estamos en modo pseudodistribuido o distribuido y queremos guardarlo en local, entonces hay que poner file:// antes de la ubicación en la que queremos guardarlo

saveAsTextFile en local

```
miRDD = sc.parallelize([4, 2, 7,
                        15, 4, 2,
                        3, 16, 3],3)

miRDD.saveAsTextFile("file:///home/hadmin/datasetGuardado")
```

countByKey

Cuenta el número de temperaturas que tenemos por cada año

```
miRDD = sc.parallelize([("1999", 7),("1999", 5), ("2000", 10),
                        ("1999", 4), ("2000", 3), ("2000", 7),
                        ("2000", 10), ("2001", 3), ("1999", 5)], 3)

salidaDriver = miRDD.countByKey()

print("\nmiRDD:", miRDD.collect(),"\nSalida en driver:", salidaDriver,
"\n")
```

foreach

Imprimir la longitud de cada registro

```
def miFuncion(record):
      print len(record)

miRDD = sc.parallelize(["uno", "dos", "tres",
                        "cuatro", "cinco", "seis",
                        "siete", "ocho", "nueve"],3)

miRDD.foreach(miFuncion)

print("\nmiRDD:", miRDD.collect())
```

getNumPartitions

Obtiene el número de particiones

```
miRDD = sc.parallelize(["uno", "dos", "tres",
                        "cuatro", "cinco", "seis",
                        "siete", "ocho", "nueve"],3)

salidaDriver = miRDD.getNumPartitions()

print("\nmiRDD:", miRDD.collect(),"\nSalida en driver:", salidaDriver,
"\n")
```

Consultar particiones

```
miRDD = sc.parallelize(["uno", "dos", "tres",
                        "cuatro", "cinco", "seis",
                        "siete", "ocho", "nueve"],3)

#imprimir particiones de miRDD
print("\nmiRDD:")
partitions = miRDD.glom().collect()
for partition in partitions:
      print("Particion: ", partition)
```