**ASSIGNMENT**

**TRANSFORMATION & ACTIONS IN PYSPARK**

**Name:Tejaswini Gokanakonda**

## **Roll no**:**DE142**

## **Date:20-11-2024**

**CODE:**

### **1. Creating RDDs and DataFrames in PySpark**

#to create rdds and  dataframe

#

from pyspark import SparkContext

from pyspark.sql import  SparkSession

sc =SparkContext.getOrCreate()

spark = SparkSession.builder.appName('pyspark first program').getOrCreate()

#create the rdd

rdd = sc.parallelize([('C',85,76,87,91), ('B',85,76,87,91), ("A", 85,78,96,92), ("A", 92,76,89,96)], 4)

mydata = ['Division','English','Mathematics','Physics','Chemistry']

marks\_df = spark.createDataFrame(rdd, schema=mydata)

print(rdd.collect())

print(rdd) #---Transformation which gives rdd value

rdd.collect() #----Action gives non rdd value

### 

### **Output Screenshot:**



### **Summary:**

### This code demonstrates the process of creating an **RDD** (Resilient Distributed Dataset) from a list of tuples and converting it into a **DataFrame** in PySpark. The RDD is created using sc.parallelize(), which takes a list of tuples, and each tuple represents data (in this case, student marks in different subjects). The DataFrame is created using spark.createDataFrame(), and the schema (column names) is provided via a list.

The code highlights the difference between **transformations** (like converting an RDD to a DataFrame) and **actions** (such as collect() which triggers the execution and retrieves the data). The collect() method is used here as an action to gather and print all the elements from the RDD, while simply printing the RDD object demonstrates how transformations are not immediately evaluated but set up a series of operations.

### **2. Creating RDDs and DataFrames in PySpark**

#to create rdds and  dataframe

#

from pyspark import SparkContext

from pyspark.sql import  SparkSession

sc =SparkContext.getOrCreate()

spark = SparkSession.builder.appName('pyspark first program').getOrCreate()

#create the rdd

rdd = sc.parallelize([('C',85,76,87,91), ('B',85,76,87,91), ("A", 85,78,96,92), ("A", 92,76,89,96)], 4)

mydata = ['Division','English','Mathematics','Physics','Chemistry']

marks\_df = spark.createDataFrame(rdd, schema=mydata)

print(rdd.count())

rdd.take(2) ##Action gives non rdd value

### **Output Screenshot:**



**Summary:**

This code demonstrates the creation of an **RDD** and a **DataFrame** in PySpark. An **RDD** is created using sc.parallelize(), which takes a list of tuples representing student marks in different subjects. The **DataFrame** is created using spark.createDataFrame(), with the schema defined by the mydata list containing column names such as 'Division', 'English', 'Mathematics', 'Physics', and 'Chemistry'.

The code performs two **actions**:

1. count(): This action counts the number of elements in the RDD and prints the total count.
2. take(2): This action retrieves the first two elements from the RDD and returns them as a list.

This example shows the distinction between **transformations** (like creating DataFrames) and **actions** (like count() and take()), where transformations do not trigger immediate computation but actions return actual data.

**3. Creating RDD and Performing Count in PySpark**

from pyspark import SparkContext

sc = SparkContext.getOrCreate()

count\_rdd = sc.parallelize([1,2,3,4,5,5,6,7,8,9])

print(count\_rdd.count())

count\_rdd.count()

### **Output Screenshot:**



**Summary:**

This code demonstrates the creation of an **RDD** using sc.parallelize() from a list of integers. The **RDD** is then processed using the count() method, which is an **action** in PySpark that returns the number of elements in the RDD. The result is printed, and the count() action is called again to show how it works.

This example highlights the **action** operation count(), which triggers computation to calculate and return the number of elements in the RDD.

4. **Creating RDD and Using Actions in PySpark**

from pyspark import SparkContext

sc = SparkContext.getOrCreate()

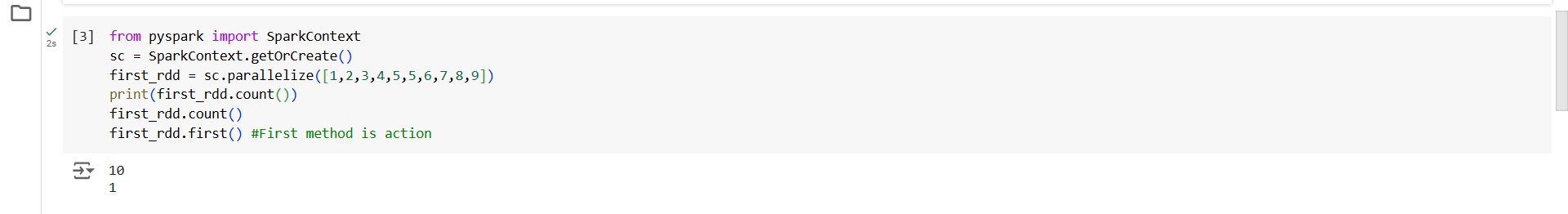
first\_rdd = sc.parallelize([1,2,3,4,5,5,6,7,8,9])

print(first\_rdd.count())

first\_rdd.count()

first\_rdd.first() #First method is action

**Output Screenshot:**

****

**Summary:**

This code demonstrates how to create an **RDD** using sc.parallelize() from a list of integers. The **RDD** is then processed with two **actions**:

1. count(): This action returns the number of elements in the RDD and prints the result.
2. first(): This action retrieves and returns the first element from the RDD.

The code shows how these **actions** trigger computation in PySpark, with count() calculating the total number of elements and first() fetching the first element of the RDD. Both actions perform actual computation and return values, highlighting the difference between **actions** and **transformations** in PySpark.

### **5. Filtering RDD in PySpark**

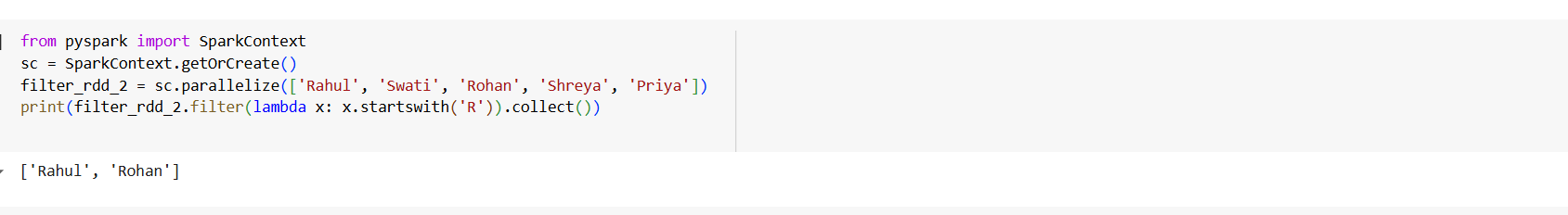
from pyspark import SparkContext

sc = SparkContext.getOrCreate()

filter\_rdd\_2 = sc.parallelize(['Rahul', 'Swati', 'Rohan', 'Shreya', 'Priya'])

print(filter\_rdd\_2.filter(lambda x: x.startswith('R')).collect())

**Output Screenshot:**

****

**Summary:**

This code demonstrates how to filter an **RDD** using the filter() transformation in PySpark. The **RDD** is created from a list of names, and the filter() method is applied to select only the names that start with the letter 'R'. The collect() action is then used to retrieve the filtered results and print them.

This example showcases the **filter transformation** to perform data selection based on a condition, and the **collect action** to retrieve and display the filtered results from the RDD.

### **6. Union of RDDs in PySpark**

from pyspark import SparkContext

sc = SparkContext.getOrCreate()

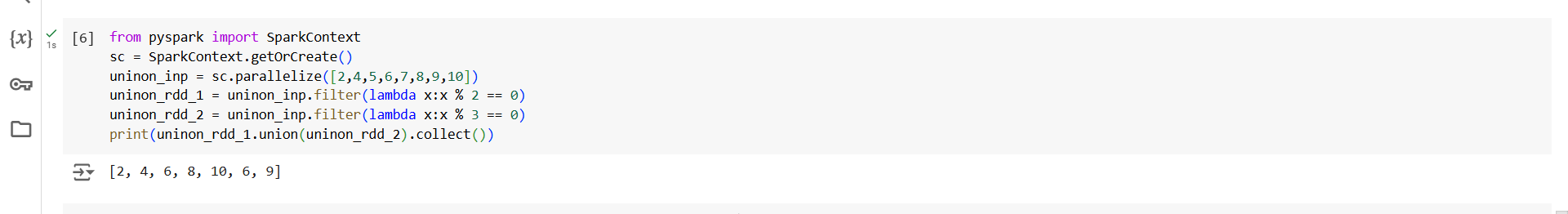
uninon\_inp = sc.parallelize([2,4,5,6,7,8,9,10])

uninon\_rdd\_1 = uninon\_inp.filter(lambda x:x % 2 == 0)

uninon\_rdd\_2 = uninon\_inp.filter(lambda x:x % 3 == 0)

print(uninon\_rdd\_1.union(uninon\_rdd\_2).collect())

**Output Screenshot:**

****

**Summary:**

This code demonstrates how to apply the **union operation** on two RDDs in PySpark. The **RDD** uninon\_inp is created from a list of integers. Two filtered RDDs are generated using the filter() method:

* uninon\_rdd\_1 contains elements that are divisible by 2.
* uninon\_rdd\_2 contains elements that are divisible by 3.

The union() transformation is then applied to combine the elements of uninon\_rdd\_1 and uninon\_rdd\_2, resulting in a new RDD that contains all the elements from both RDDs. The collect() action is used to retrieve and print the unioned result.

This example illustrates the use of the **union transformation** to merge two RDDs, where the union includes all unique elements from both, and the **collect action** to retrieve and display the final result.

### **7. FlatMap Transformation in PySpark**

from pyspark import SparkContext

sc = SparkContext.getOrCreate()

flatmap\_rdd = sc.parallelize(["Hey there", "This is PySpark RDD Transformations"])

print(flatmap\_rdd.flatMap(lambda x :x.split(" ").collect()))

flatmap\_rdd.flatMap(lambda x :x.split(" ").collect())

**Output Screenshot:**

****

**Summary:**

This code demonstrates the use of the **flatMap transformation** in PySpark. The **RDD** flatmap\_rdd is created from a list of strings. The flatMap() method is applied, which splits each string into words using the split(" ") function. Unlike map(), which would return an RDD of lists, flatMap() flattens the results, resulting in an RDD of individual words.

The collect() action is used to retrieve and print the final result of the transformation.

This example highlights the **flatMap transformation**, which is useful when you want to apply a transformation that outputs multiple values for each element, flattening the result into a single RDD, and then using the **collect action** to retrieve the results.

### **8. FlatMap Transformation in PySpark**

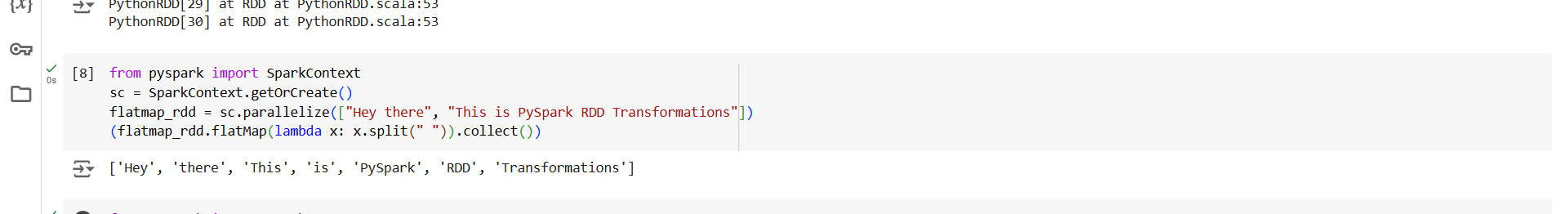
from pyspark import SparkContext

sc = SparkContext.getOrCreate()

flatmap\_rdd = sc.parallelize(["Hey there", "This is PySpark RDD Transformations"])

(flatmap\_rdd.flatMap(lambda x: x.split(" ")).collect())

**Output Screenshot:**

****

**Summary:**

This code demonstrates the use of the **flatMap transformation** in PySpark. The **RDD** flatmap\_rdd is created from a list of strings. The flatMap() transformation is applied, which splits each string into individual words using the split(" ") method. Unlike map(), which would return an RDD of lists, flatMap() produces a flattened RDD containing all the words as individual elements.

The collect() action is used to retrieve and display the resulting RDD of words.

This example illustrates how **flatMap** works to transform and flatten the output, creating an RDD of individual words from multiple strings, and how **collect()** is used to retrieve the result.

### **9. Reduce Transformation in PySpark**

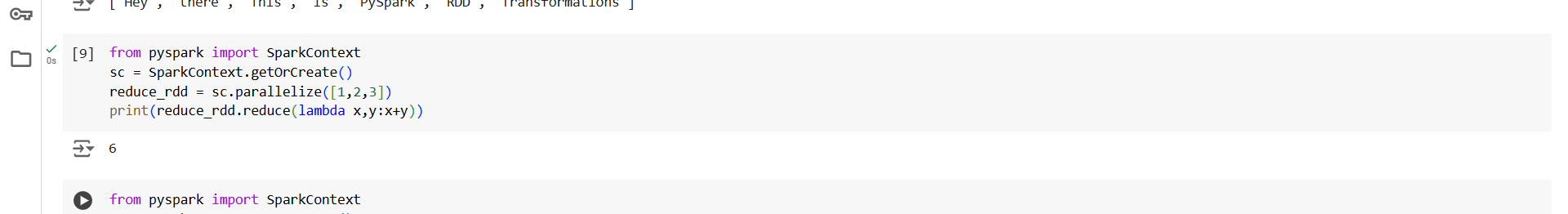
from pyspark import SparkContext

sc = SparkContext.getOrCreate()

reduce\_rdd = sc.parallelize([1,2,3])

print(reduce\_rdd.reduce(lambda x,y:x+y))

**Output Screenshot:**

****

**Summary:**

This code demonstrates the use of the **reduce transformation** in PySpark. The **RDD** reduce\_rdd is created from a list of integers. The reduce() method is applied with a lambda function that adds two values at a time, combining them into a single result. The operation is performed iteratively on all elements in the RDD to produce a final aggregated result.

In this case, the code reduces the elements of the RDD by summing them, and the final result is printed.

This example illustrates the **reduce transformation**, which is used to aggregate or combine the elements of an RDD into a single value based on the provided operation (in this case, addition). The result is computed using **reduction** across the RDD elements.

### **10. Map Transformation in PySpark**

from pyspark import SparkContext

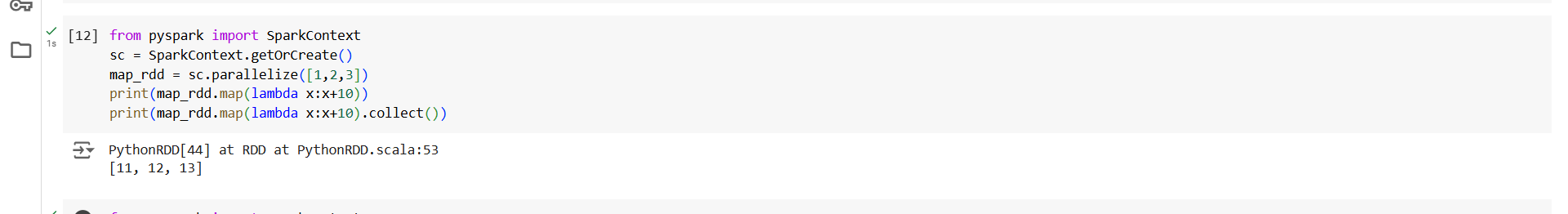
sc = SparkContext.getOrCreate()

map\_rdd = sc.parallelize([1,2,3])

print(map\_rdd.map(lambda x:x+10))

print(map\_rdd.map(lambda x:x+10).collect())

**Output Screenshot:**



**Summary:**

This code demonstrates the use of the **map transformation** in PySpark. The **RDD** map\_rdd is created from a list of integers. The map() method is applied with a lambda function that adds 10 to each element in the RDD. The transformation results in a new RDD where each value is increased by 10.

* The first print() statement will output the transformed RDD, but since it hasn't been collected, it won't show the actual values yet.
* The second print() statement uses the collect() action to retrieve and display the final result of the transformation, which contains the elements after the addition.

This example demonstrates how the **map transformation** applies a function (in this case, adding 10) to each element in the RDD, and how the **collect action** is used to retrieve and display the transformed data.

### **11. Filter Transformation in PySpark**

from pyspark import SparkContext

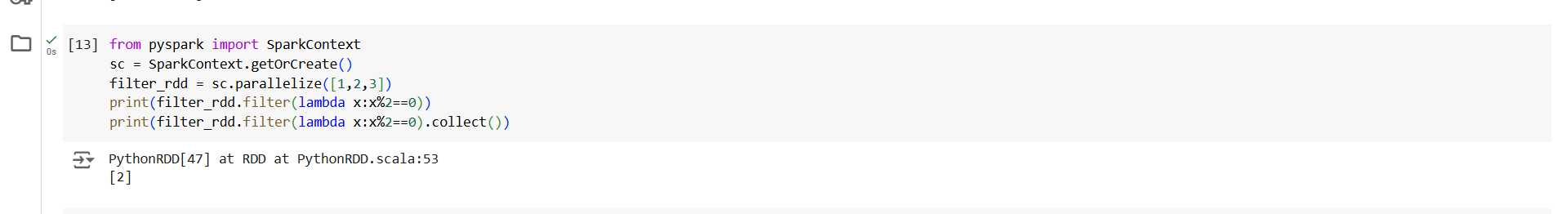
sc = SparkContext.getOrCreate()

filter\_rdd = sc.parallelize([1,2,3])

print(filter\_rdd.filter(lambda x:x%2==0))

print(filter\_rdd.filter(lambda x:x%2==0).collect())

### **Output Screenshot:**



### **Summary:**

This code demonstrates the use of the **filter transformation** in PySpark. The **RDD** filter\_rdd is created from a list of integers. The filter() method is applied with a lambda function that selects only the even numbers from the RDD (x % 2 == 0).

* The first print() statement will output the filtered RDD object, but since it hasn't been collected, it won't display the actual filtered data.
* The second print() statement uses the collect() action to retrieve and display the final result, which contains only the even numbers.

This example shows how the **filter transformation** works to select specific elements (in this case, even numbers) based on a condition, and how the **collect action** is used to retrieve and display the results of the transformation.

### **12. Take Action in PySpark**

from pyspark import SparkContext

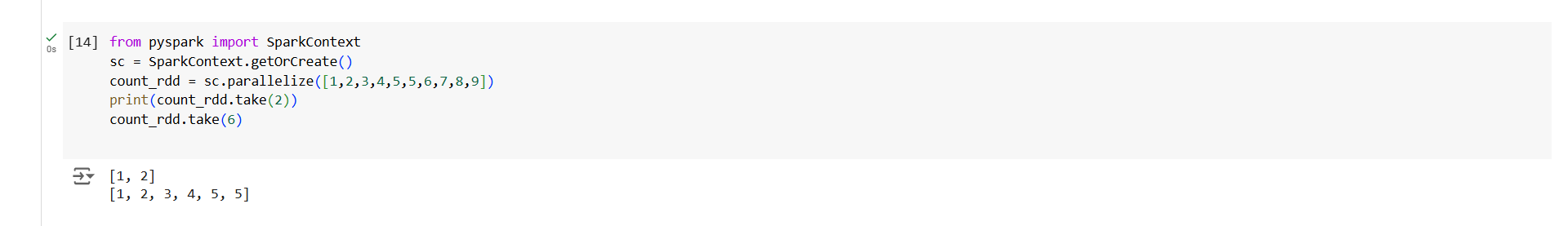
sc = SparkContext.getOrCreate()

count\_rdd = sc.parallelize([1,2,3,4,5,5,6,7,8,9])

print(count\_rdd.take(2))

count\_rdd.take(6)

**Output Screenshot:**



**Summary:**

This code demonstrates the use of the **take action** in PySpark. The **RDD** count\_rdd is created from a list of integers. The take() method is applied to retrieve the first few elements from the RDD.

* The first print() statement uses take(2), which retrieves the first 2 elements of the RDD.
* The second take(6) retrieves the first 6 elements of the RDD, but the result is not printed in the code.

This example shows how the **take action** is used to fetch a specified number of elements from the beginning of the RDD. It’s particularly useful when you want to inspect or work with a small portion of a large RDD.