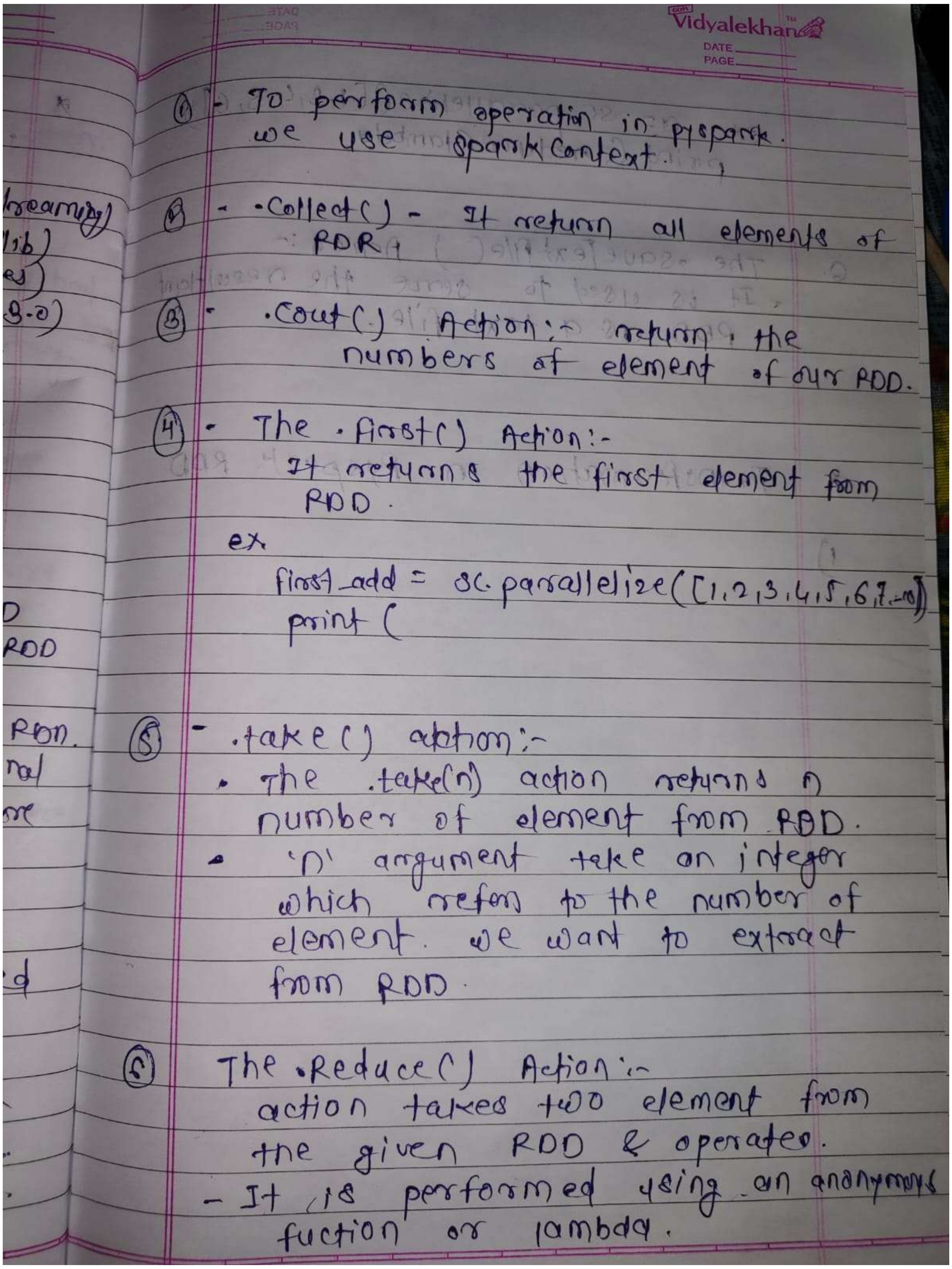
Day 13 Assessment

Name: Krushnakumar Patle

Email: krishnapatle128@gmail.com

Batch: Data Engineering Batch-1



Vidyalekhan

DATE

PAGE

① - To perform operation in pyspark. we use spark Context.

② - .Collect() - It return all elements of RDR.

③ - .Cout() Action:- return the numbers of element of our RDD.

④ - The .first() Action:-
It returns the first element from RDD.

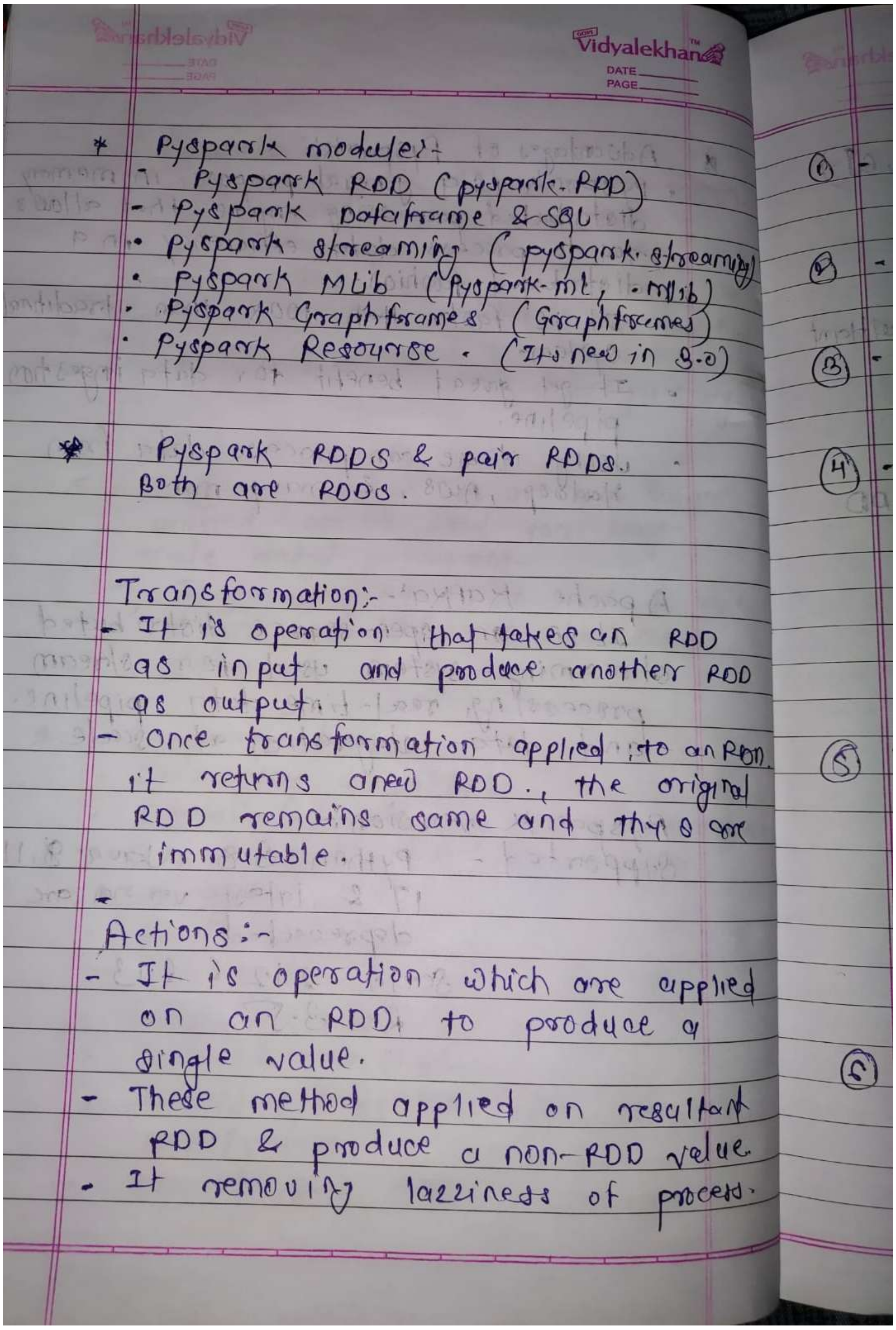
ex

```
first_add = sc.parallelize([1,2,3,4,5,6,7...10])
print (
```

⑤ - .take() action:-
• The .take(n) action returns n number of element from RDD.
• 'n' argument take an integer which refers to the number of element. we want to extract from RDD.

⑥ The .Reduce() Action:-
action takes two element from the given RDD & operates.
- It is performed using an anonymous fuction or lambda.

Day 13 Assessment



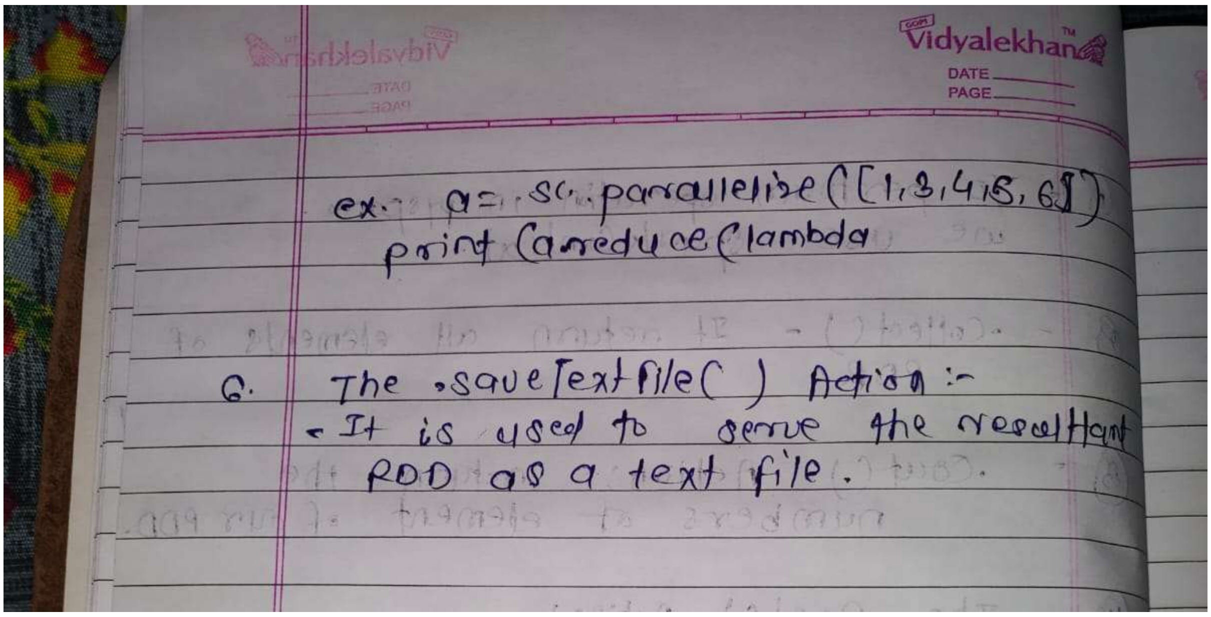
* Pyspark module:-
- Pyspark RDD (pyspark.RDD)
- Pyspark Dataframe & SQL
• Pyspark streaming (pyspark.streaming)
• Pyspark MLib (pyspark.ml, .mllib)
• Pyspark Graphframes (Graphframes)
• Pyspark Resourse. (It's new in 3.0)

* Pyspark RDDS & pair RDDs.
Both are RDDS.

Transformation:-
- It is operation that takes an RDD as input and produce another RDD as output.
- Once transformation applied to an RDD. it returns a new RDD., the original RDD remains same and thus are immutable.
-

Actions:-
- It is operation which are applied on an RDD to produce a single value.
- These method applied on resultant RDD & produce a non-RDD value.
• It removing lazziness of process.

Day 13 Assessment



ex. a = sc.parallelize([1,3,4,5,6])
print (a.reduce(lambda

6. The .saveTextFile( ) Action :-
- It is used to serve the resultant RDD as a text file.

```
[1]: import pyspark
     import findspark
     findspark.init()
```

```
[2]: from pyspark import SparkContext
     sc = SparkContext("local", "RDD Transformation")
     sc
```

[2]: **SparkContext**

Spark UI

Version	v3.5.0
Master	local
AppName	RDD Transformation

Actions in PySpark RDDs

In PySpark RDDs, Actions are a kind of operation that returns a value on being applied to an RDD.

1) .count() Action

It returns the number of element available in RDD. Consider the following program.

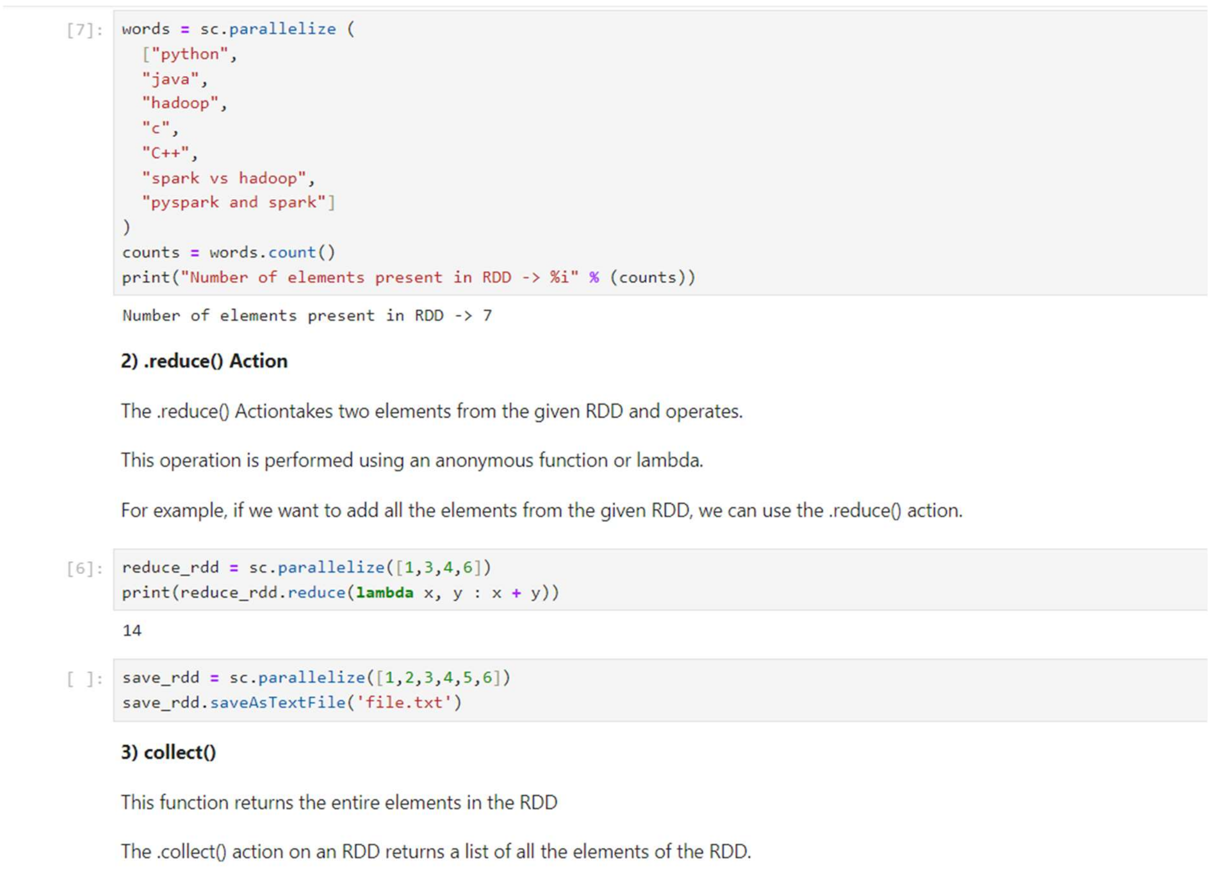
The .count() action on an RDD is an operation that returns the number of elements of our RDD.

This helps in verifying if a correct number of elements are being added in an RDD. Let's understand this with an example:

```
[3]: count_rdd = sc.parallelize([1,2,3,4,5,5,6,7,8,9])
     print(count_rdd.count())
```

```
10
```

Day 13 Assessment



```
[7]: words = sc.parallelize (
       ["python",
       "java",
       "hadoop",
       "c",
       "C++",
       "spark vs hadoop",
       "pyspark and spark"]
     )
     counts = words.count()
     print("Number of elements present in RDD -> %i" % (counts))
```

```
Number of elements present in RDD -> 7
```

2) .reduce() Action

The .reduce() Actiontakes two elements from the given RDD and operates.

This operation is performed using an anonymous function or lambda.

For example, if we want to add all the elements from the given RDD, we can use the .reduce() action.

```
[6]: reduce_rdd = sc.parallelize([1,3,4,6])
     print(reduce_rdd.reduce(lambda x, y : x + y))
```

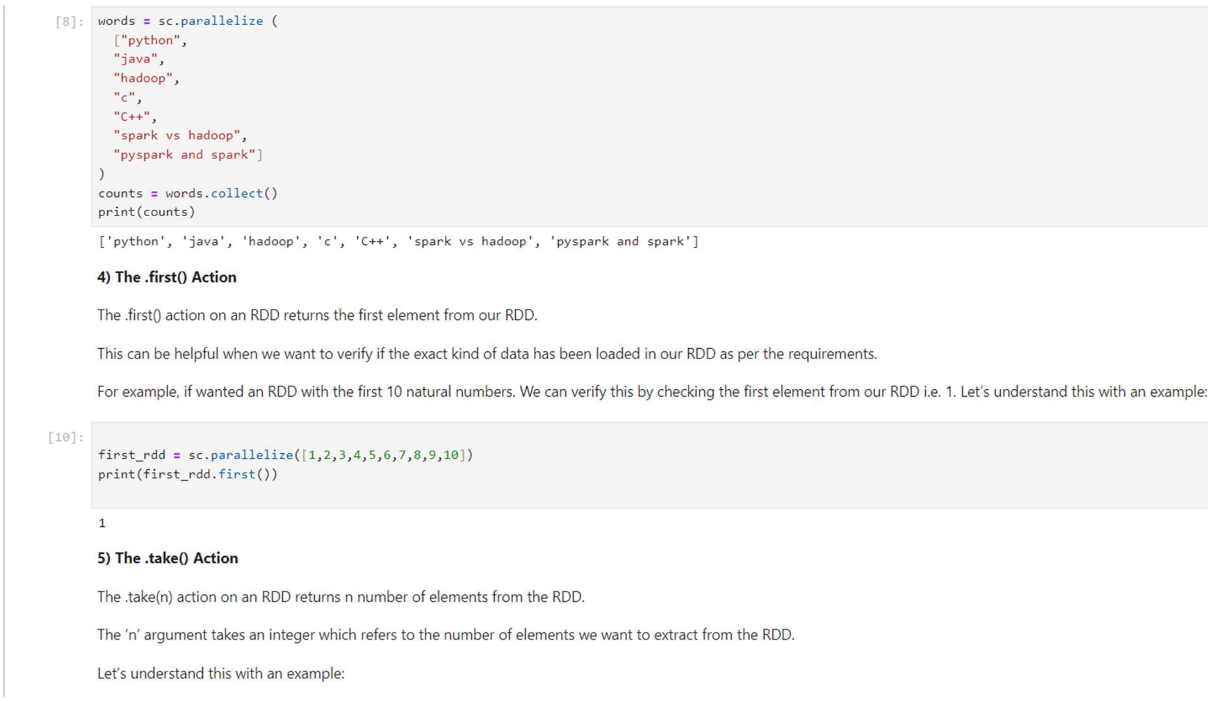
```
14
```

```
[ ]: save_rdd = sc.parallelize([1,2,3,4,5,6])
     save_rdd.saveAsTextFile('file.txt')
```

3) collect()

This function returns the entire elements in the RDD

The .collect() action on an RDD returns a list of all the elements of the RDD.



```
[8]: words = sc.parallelize (
       ["python",
       "java",
       "hadoop",
       "c",
       "C++",
       "spark vs hadoop",
       "pyspark and spark"]
     )
     counts = words.collect()
     print(counts)
```

```
['python', 'java', 'hadoop', 'c', 'C++', 'spark vs hadoop', 'pyspark and spark']
```

4) The .first() Action

The .first() action on an RDD returns the first element from our RDD.

This can be helpful when we want to verify if the exact kind of data has been loaded in our RDD as per the requirements.

For example, if wanted an RDD with the first 10 natural numbers. We can verify this by checking the first element from our RDD i.e. 1. Let's understand this with an example:

```
[10]:
     first_rdd = sc.parallelize([1,2,3,4,5,6,7,8,9,10])
     print(first_rdd.first())
```

```
1
```

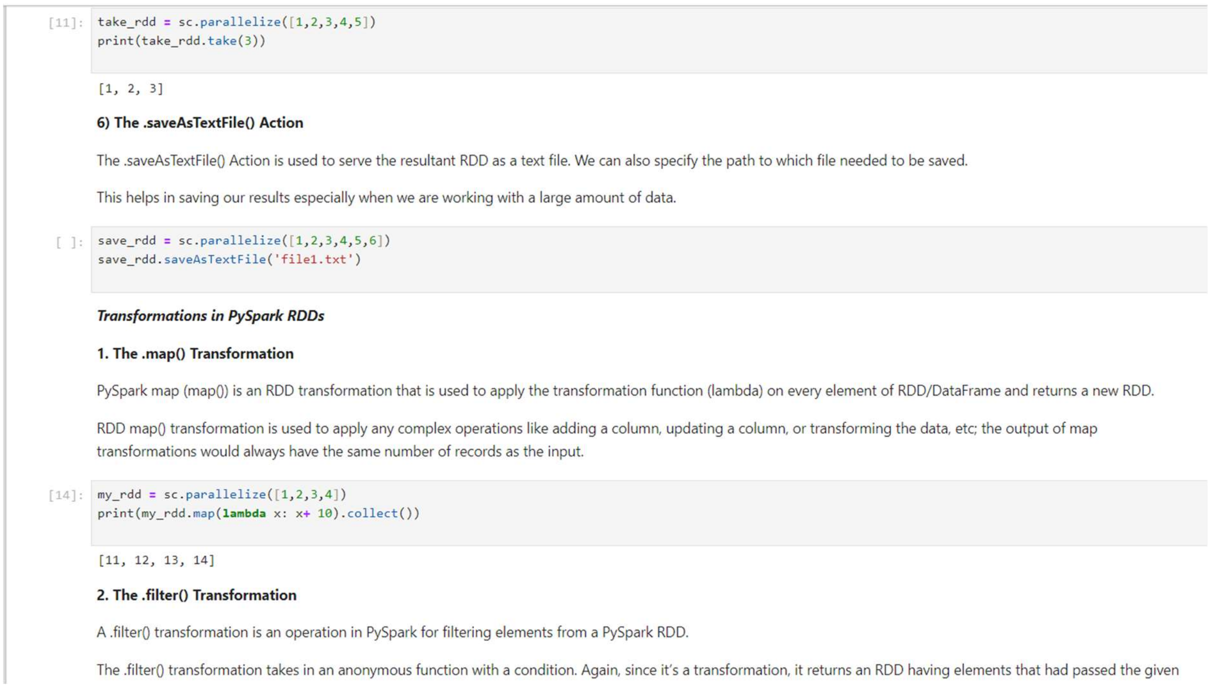
5) The .take() Action

The .take(n) action on an RDD returns n number of elements from the RDD.

The 'n' argument takes an integer which refers to the number of elements we want to extract from the RDD.

Let's understand this with an example:

Day 13 Assessment



```
[11]: take_rdd = sc.parallelize([1,2,3,4,5])
      print(take_rdd.take(3))
```

```
[1, 2, 3]
```

6) The .saveAsTextFile() Action

The .saveAsTextFile() Action is used to serve the resultant RDD as a text file. We can also specify the path to which file needed to be saved.

This helps in saving our results especially when we are working with a large amount of data.

```
[ ]: save_rdd = sc.parallelize([1,2,3,4,5,6])
     save_rdd.saveAsTextFile('file1.txt')
```

Transformations in PySpark RDDs

1. The .map() Transformation

PySpark map (map()) is an RDD transformation that is used to apply the transformation function (lambda) on every element of RDD/DataFrame and returns a new RDD.

RDD map() transformation is used to apply any complex operations like adding a column, updating a column, or transforming the data, etc; the output of map transformations would always have the same number of records as the input.

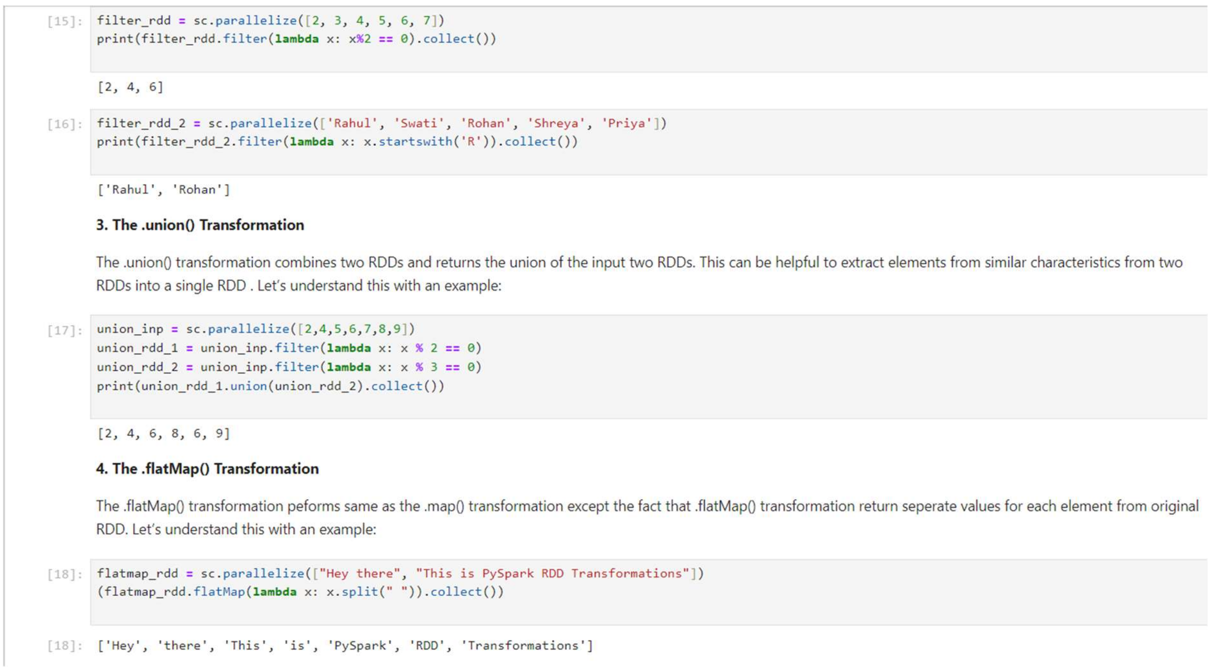
```
[14]: my_rdd = sc.parallelize([1,2,3,4])
      print(my_rdd.map(lambda x: x+ 10).collect())
```

```
[11, 12, 13, 14]
```

2. The .filter() Transformation

A .filter() transformation is an operation in PySpark for filtering elements from a PySpark RDD.

The .filter() transformation takes in an anonymous function with a condition. Again, since it's a transformation, it returns an RDD having elements that had passed the given



```
[15]: filter_rdd = sc.parallelize([2, 3, 4, 5, 6, 7])
      print(filter_rdd.filter(lambda x: x%2 == 0).collect())
```

```
[2, 4, 6]
```

```
[16]: filter_rdd_2 = sc.parallelize(['Rahul', 'Swati', 'Rohan', 'Shreya', 'Priya'])
      print(filter_rdd_2.filter(lambda x: x.startswith('R')).collect())
```

```
['Rahul', 'Rohan']
```

3. The .union() Transformation

The .union() transformation combines two RDDs and returns the union of the input two RDDs. This can be helpful to extract elements from similar characteristics from two RDDs into a single RDD . Let's understand this with an example:

```
[17]: union_inp = sc.parallelize([2,4,5,6,7,8,9])
      union_rdd_1 = union_inp.filter(lambda x: x % 2 == 0)
      union_rdd_2 = union_inp.filter(lambda x: x % 3 == 0)
      print(union_rdd_1.union(union_rdd_2).collect())
```

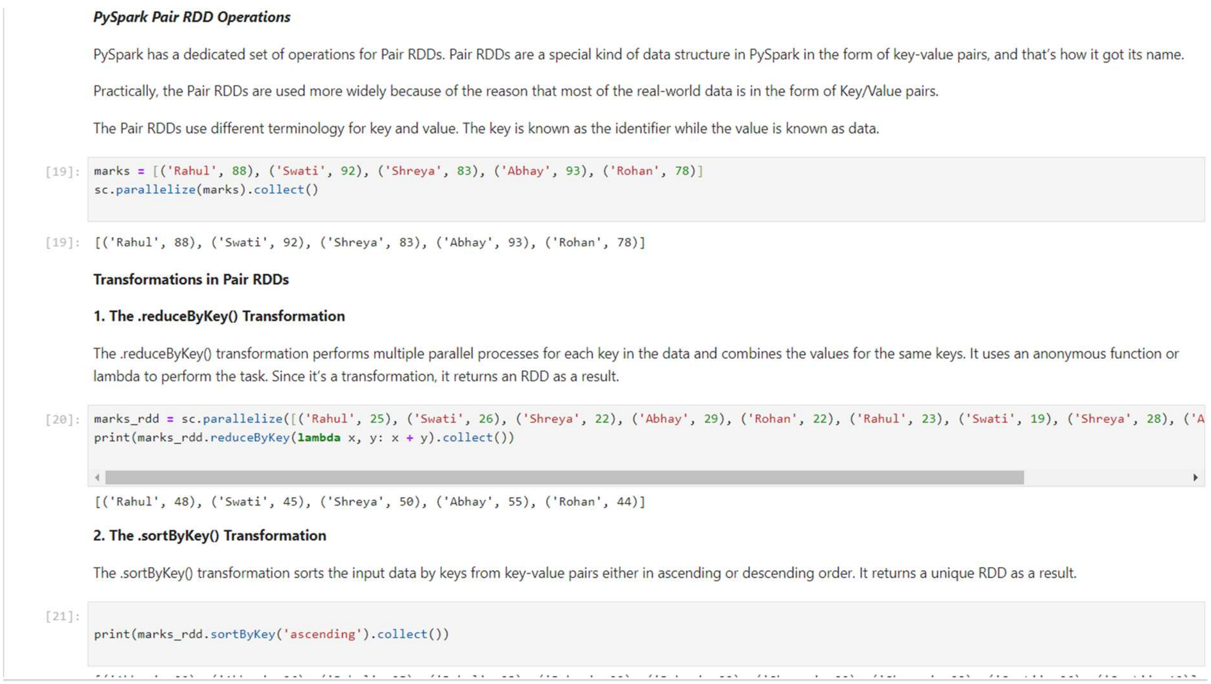
```
[2, 4, 6, 8, 6, 9]
```

4. The .flatMap() Transformation

The .flatMap() transformation peforms same as the .map() transformation except the fact that .flatMap() transformation return seperate values for each element from original RDD. Let's understand this with an example:

```
[18]: flatmap_rdd = sc.parallelize(["Hey there", "This is PySpark RDD Transformations"])
      (flatmap_rdd.flatMap(lambda x: x.split(" ")).collect())
```

```
[18]: ['Hey', 'there', 'This', 'is', 'PySpark', 'RDD', 'Transformations']
```



PySpark Pair RDD Operations

PySpark has a dedicated set of operations for Pair RDDs. Pair RDDs are a special kind of data structure in PySpark in the form of key-value pairs, and that's how it got its name.

Practically, the Pair RDDs are used more widely because of the reason that most of the real-world data is in the form of Key/Value pairs.

The Pair RDDs use different terminology for key and value. The key is known as the identifier while the value is known as data.

```
[19]: marks = [('Rahul', 88), ('Swati', 92), ('Shreya', 83), ('Abhay', 93), ('Rohan', 78)]
      sc.parallelize(marks).collect()

[19]: [('Rahul', 88), ('Swati', 92), ('Shreya', 83), ('Abhay', 93), ('Rohan', 78)]
```

Transformations in Pair RDDs

1. The .reduceByKey() Transformation

The .reduceByKey() transformation performs multiple parallel processes for each key in the data and combines the values for the same keys. It uses an anonymous function or lambda to perform the task. Since it's a transformation, it returns an RDD as a result.

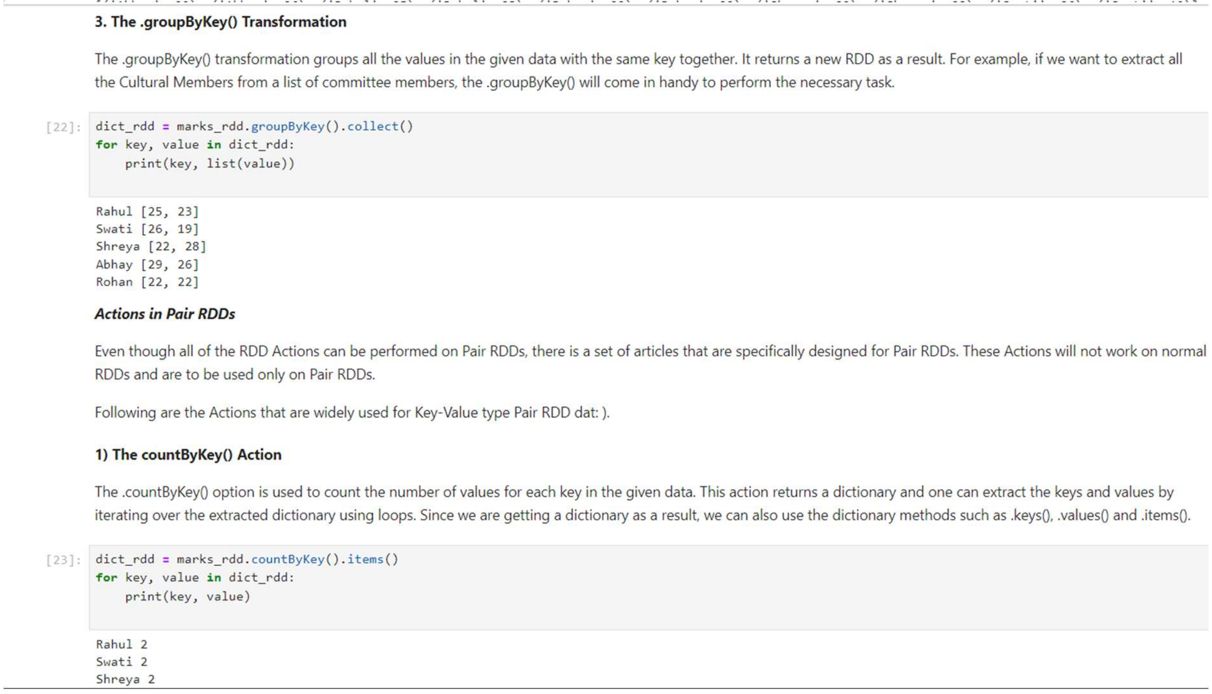
```
[20]: marks_rdd = sc.parallelize([('Rahul', 25), ('Swati', 26), ('Shreya', 22), ('Abhay', 29), ('Rohan', 22), ('Rahul', 23), ('Swati', 19), ('Shreya', 28), ('A
      print(marks_rdd.reduceByKey(lambda x, y: x + y).collect())

      [('Rahul', 48), ('Swati', 45), ('Shreya', 50), ('Abhay', 55), ('Rohan', 44)]
```

2. The .sortByKey() Transformation

The .sortByKey() transformation sorts the input data by keys from key-value pairs either in ascending or descending order. It returns a unique RDD as a result.

```
[21]:
      print(marks_rdd.sortByKey('ascending').collect())
```



3. The .groupByKey() Transformation

The .groupByKey() transformation groups all the values in the given data with the same key together. It returns a new RDD as a result. For example, if we want to extract all the Cultural Members from a list of committee members, the .groupByKey() will come in handy to perform the necessary task.

```
[22]: dict_rdd = marks_rdd.groupByKey().collect()
      for key, value in dict_rdd:
          print(key, list(value))

      Rahul [25, 23]
      Swati [26, 19]
      Shreya [22, 28]
      Abhay [29, 26]
      Rohan [22, 22]
```

Actions in Pair RDDs

Even though all of the RDD Actions can be performed on Pair RDDs, there is a set of articles that are specifically designed for Pair RDDs. These Actions will not work on normal RDDs and are to be used only on Pair RDDs.

Following are the Actions that are widely used for Key-Value type Pair RDD dat:).

1) The countByKey() Action

The .countByKey() option is used to count the number of values for each key in the given data. This action returns a dictionary and one can extract the keys and values by iterating over the extracted dictionary using loops. Since we are getting a dictionary as a result, we can also use the dictionary methods such as .keys(), .values() and .items().

```
[23]: dict_rdd = marks_rdd.countByKey().items()
      for key, value in dict_rdd:
          print(key, value)

      Rahul 2
      Swati 2
      Shreya 2
```

DataFrames in Pyspark:

In PySpark, DataFrames are distributed collections of data organized into named columns. They are similar to pandas DataFrames but are designed to handle large-scale data processing in distributed environments. You can create DataFrames from various data sources, including structured data files, databases, or existing RDDs (Resilient Distributed Datasets).

Day 13 Assessment

[1]:
```
import os
import sys

os.environ["PYSPARK_PYTHON"]=sys.executable
os.environ["PYSPARK_DRIVER_PYTHON"]=sys.executable
```

[2]:
```
import pyspark
import findspark
findspark.init()
```

[3]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('DataFrames').getOrCreate()
```

[4]:
```
# Create data in dataframe
data = [(('Ram'), '1991-04-01', 'M', 3000),
        (('Mike'), '2000-05-19', 'M', 4000),
        (('Rohini'), '1978-09-05', 'M', 4000),
        (('Maria'), '1967-12-01', 'F', 4000),
        (('Jenis'), '1980-02-17', 'F', 1200)]

# Column names in dataframe
columns = ["Name", "DOB", "Gender", "salary"]

# Create the spark dataframe
df = spark.createDataFrame(data=data,
                           schema=columns)

# Print the dataframe
df.show()
```

```
+------+----------+------+------+
|  Name|       DOB|Gender|salary|
+------+----------+------+------+
|   Ram|1991-04-01|     M|  3000|
|  Mike|2000-05-19|     M|  4000|
|Rohini|1978-09-05|     M|  4000|
| Maria|1967-12-01|     F|  4000|
| Jenis|1980-02-17|     F|  1200|
+------+----------+------+------+
```

Method 1: Using withColumnRenamed()

We will use of withColumnRenamed() method to change the column names of pyspark data frame.

[5]:
```
df.withColumnRenamed("DOB","DateOfBirth").show()
```

```
+------+-----------+------+------+
|  Name|DateOfBirth|Gender|salary|
+------+-----------+------+------+
|   Ram| 1991-04-01|     M|  3000|
|  Mike| 2000-05-19|     M|  4000|
|Rohini| 1978-09-05|     M|  4000|
| Maria| 1967-12-01|     F|  4000|
| Jenis| 1980-02-17|     F|  1200|
+------+-----------+------+------+
```

[7]:
```
df.withColumnRenamed("Gender","Sex").withColumnRenamed("salary","Amount").show()
```

```
+------+----------+---+------+
|  Name|       DOB|Sex|Amount|
+------+----------+---+------+
|   Ram|1991-04-01|  M|  3000|
|  Mike|2000-05-19|  M|  4000|
|Rohini|1978-09-05|  M|  4000|
| Maria|1967-12-01|  F|  4000|
| Jenis|1980-02-17|  F|  1200|
+------+----------+---+------+
```

Day 13 Assessment

```
[7]: df.withColumnRenamed("Gender","Sex").withColumnRenamed("salary","Amount").show()
```

```
+------+----------+---+------+
|  Name|       DOB|Sex|Amount|
+------+----------+---+------+
|   Ram|1991-04-01|  M|  3000|
|  Mike|2000-05-19|  M|  4000|
|Rohini|1978-09-05|  M|  4000|
| Maria|1967-12-01|  F|  4000|
| Jenis|1980-02-17|  F|  1200|
+------+----------+---+------+
```

Method 2: Using selectExpr()

Renaming the column names using selectExpr() method

```
[8]: # Select the 'Name' as 'name'
     # Select remaining with their original name
     data = df.selectExpr("Name as name","DOB","Gender","salary")

     # Print the dataframe
     data.show()
```

```
+------+----------+------+------+
|  name|       DOB|Gender|salary|
+------+----------+------+------+
|   Ram|1991-04-01|     M|  3000|
|  Mike|2000-05-19|     M|  4000|
|Rohini|1978-09-05|     M|  4000|
| Maria|1967-12-01|     F|  4000|
| Jenis|1980-02-17|     F|  1200|
+------+----------+------+------+
```

Method 3: Using select() methodunt'

[9]:
```
from pyspark.sql.functions import col

# Select the 'salary' as 'Amount' using aliasing
# Select remaining with their original name
data = df.select(col("Name"),col("DOB"),
                 col("Gender"),
                 col("salary").alias('Amount'))
# Print the dataframe
data.show()
```

```
+------+----------+------+------+
|  Name|       DOB|Gender|Amount|
+------+----------+------+------+
|   Ram|1991-04-01|     M|  3000|
|  Mike|2000-05-19|     M|  4000|
|Rohini|1978-09-05|     M|  4000|
| Maria|1967-12-01|     F|  4000|
| Jenis|1980-02-17|     F|  1200|
+------+----------+------+------+
```

Method 4: Using toDF()

This function returns a new DataFrame that with new specified column names.

[10]:
```
Data_list = ["Emp Name","Date of Birth",
             " Gender-m/f","Paid salary"]

new_df = df.toDF(*Data_list)
new_df.show()
```

```
+--------+-------------+-----------+-----------+
|Emp Name|Date of Birth| Gender-m/f|Paid salary|
+--------+-------------+-----------+-----------+
|     Ram|   1991-04-01|          M|       3000|
|    Mike|   2000-05-19|          M|       4000|
|  Rohini|   1978-09-05|          M|       4000|
|   Maria|   1967-12-01|          F|       4000|
|   Jenis|   1980-02-17|          F|       1200|
+--------+-------------+-----------+-----------+
```