Experiment-04

1) Spark frame work is from apache vendor which is open source
2) spark is only meant for big data processing (using In-memory cluster computing primitive) ---however it's not meant for storage.
3) spark does not have any storage component --- and it can read the input data from local file system (LFS). HDFS, RDBMS, NoSQL, cloud vendors (Azure / AWS).

→ One of the processing is done from spark, it can write the o/p data to any of the above storage systems.

→ When we install spark we will see (spark core, spark sql, spark streaming, sparkMLlib, spark Graph-X)
5 modules

→ Spark is a unified stack

→ spark processing can be achieved through
1. scala language
2. Python (pyspark
3. R language
4. Java language

} four compatable

<u>Advantage</u>:-

The main advantage of spark is because of its In-memory cluster computing & processing which is not present in our hadoop, hadoop is a desk based-processing system.

→ Hadoop follows <u>top to bottom</u> approach where as spark follows <u>bottom to top</u> approach.

→ If we want to see the different methods in spark, we need to type scala> sc. (And press tab)
It will show the different methods in spark.

→ In spark we have 2 different RDD operations like
① Transformation ② Action

1) Transformation is a method of an RDD creates a new RDD by performing a computation on the source RDD

1) Map
2) Filter
3) flat map
4) Map partitions
5) Union
6) Intersection
7) substract
8) sort by

2) Actions are RDD methods that return a value to a driver.

① collect - method returns elements in source RDD as an array.

② count - The method returns a count of elements in source RDD.

→ To run spark processing in a standalone cluster manage node

1) Using scala → Spark-shell
2) Using python → pyspark
3) Using R → sparkR
4) using java → No support

```
I scala> var file =sc.textFile("file:///home/vboxuser/prac/spark/
                                        Input.log")
scala> var mapfile = file.map(x=>(x,x.length))
scala> mapfile.collect.foreach(println)
output:- (Apache spark is an open source, ,32)
  (It is wide range data processing engine, 38)
  (There is a common belief that Apache spark ,42)
  (Is an extension of Hadoop ,28)
  (It uses Hadoop for storage purpose only. ,41)
II scala> var data =sc.parallelize(List("spark","scala","c",
     "cobol", "vc++", "Hadoop", "Ruby", "information", "blockchain",
  "generativeAI", "chat Gpt", "java", "python", "c++"))
scala> var mapdata = data.map(x => (x,x.length))
scala> Mapdata. collect.foreach (println)
```

```
output:- (spark,5)
         (scala,5)
         (c,1)
         (cobol,5)
         (vc++,4)
         (hadoop,6)
         (ruby,4)
         (information,11)
         (blockchain,10)
         (generativeAI,12)
         (chatGPT,7)
         (java,4)
         (python,6)
         (c++,3)
```

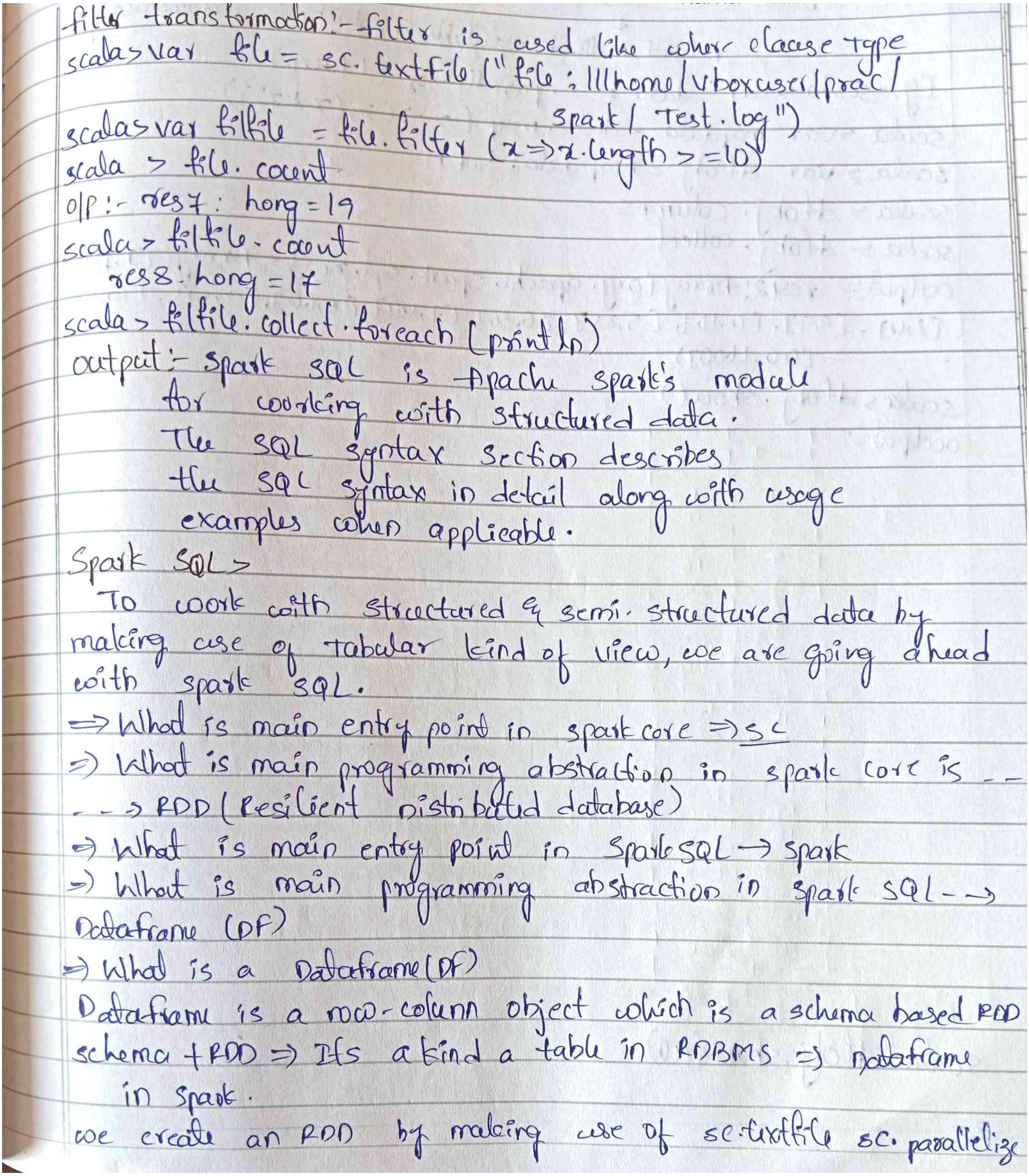
→ Here collect gathers all the elements of RDD or dataframe.

→ for each is an action that implies a function to each element of collected data.

→ println is simply printing out each element

→ It collects all elements from Mapdata & then prints each tuple to the console.

```
scala> var Mapdata = data.map (x => (x, x.reverse, x.length))
scala> mapdata. collect . foreach (println)
output:- (spark, kraps, 5)
         (scala, alacs, 5)
         (c,c,1)
         (cobol, loboc,5)
         (VC++,++CV,4)
         (hadoop, poodah,6)
         (Ruby, ybuR,4)
         (information,noitamrofni, 11)
         (blockchain,niahckcolb,10)
         (generativeAI,IA evitareneg,12)
         (chatGPT,TPG tahc, 7)
         (java, avaj,4)
         (python,nohtyp,6)  (c++,++c,3)
```



filter transformation:- filter is used like where clause type

```
scala> var file = sc.textFile("file:///home/vboxuser/prac/
                                   spark/Test.log")
scala> var filfile = file.filter(x=>x.length>=10)
scala> file.count
o/p:- res7: Long=19
scala> filfile.count
   res8: Long=17
scala> filfile.collect.foreach(println)
output:- Spark SQL is Apache spark's module
      for working with structured data.
      The SQL syntax section describes
      the SQL syntax in detail along with usage
      examples when applicable.
```

Spark SQL>

To work with structured & semi-structured data by making use of Tabular kind of view, we are going ahead with spark SQL.

=> What is main entry point in spark core => sc

=> What is main programming abstraction in spark core is --- -> RDD (Resilient Distributed database)

=> What is main entry point in Spark SQL -> spark

=> What is main programming abstraction in spark SQL --> Dataframe (DF)

=> What is a Dataframe (DF)

Dataframe is a row-column object which is a schema based RDD schema + RDD => It's a kind a table in RDBMS => Dataframe in spark.

we create an RDD by making use of sc.textfile sc.parallelize

how to convert RDD to Data frame.

⇒ from the created RDD convert rdd to dataframe.

syntax:- $\underbrace{\text{rdd-object.toDF}}_{}$

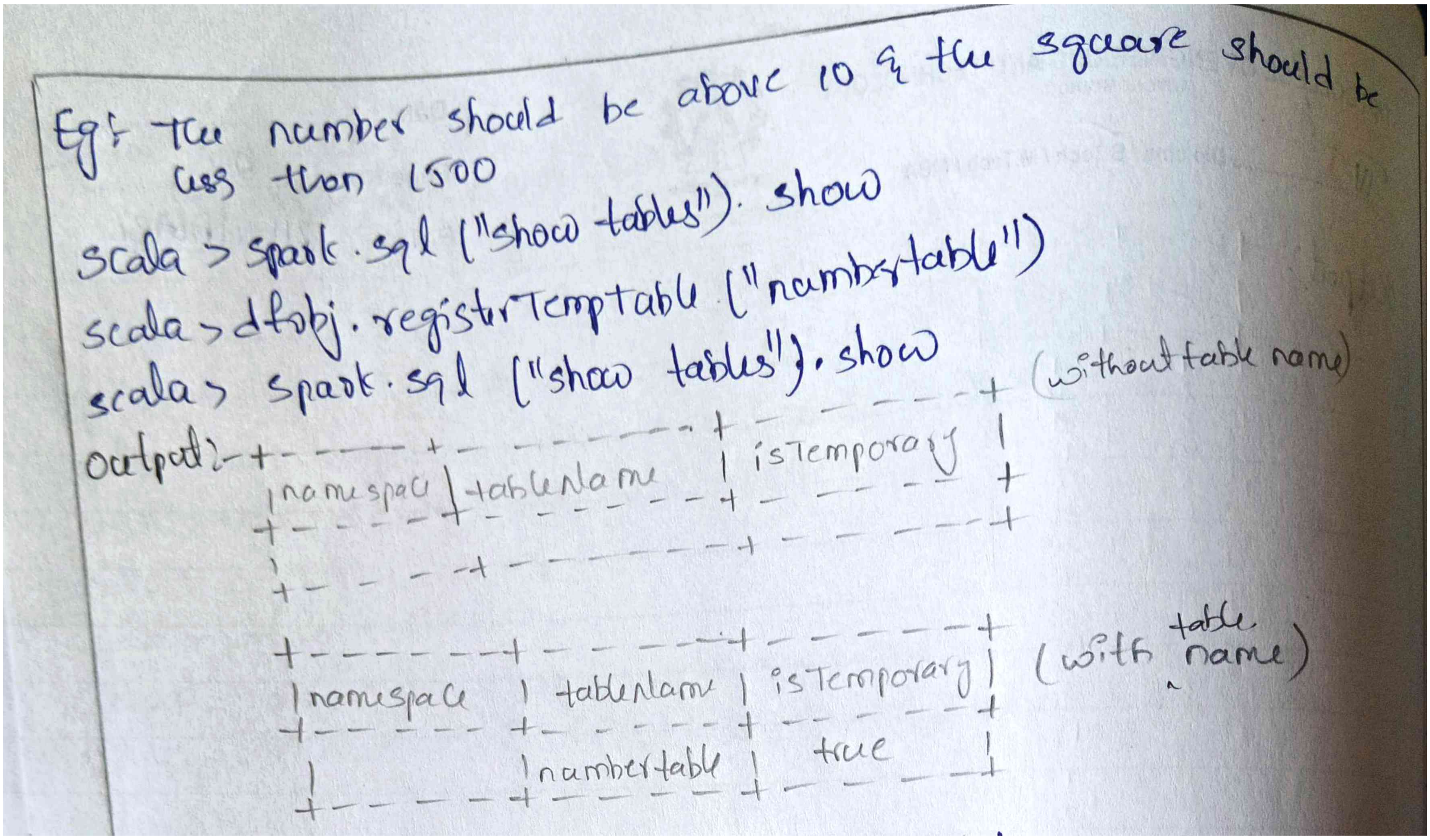
↓

Default method available in which is automatically

Eg:-
```
scala> var data = sc.parallelize (1 to 40)
scala> var mapdata = data.map (x=>(x,(x*x)))
scala> var dfobj = mapdata.toDF
scala> dfobj.columns
scala> dfobj.collect
output:- res12: Array[org.apache.spark.sql.row] = Array([1,1],
[8,64],[9,81],[10,100],
[2,4], [3,9], [4,16], [5,25], [6,36], [7,49],
------ [40,1600])
scala> dfobj.show()
output:- +---+---+
         | _1 | _2 |
         +---+---+
         | 1  |  1  |
         | 2  |  4  |
         | 3  |  9  |
         | 4  |  16 |
         | 5  |  25 |
         | 6  |  36 |
         | 7  |  49 |
         | 8  |  64 |
         | 9  |  81 |
         | 10 |  100 |
         | 11 |  121 |
         | 12 |      |
         | 13 |  144 |
         | 14 |  169 |
         | 15 |  196 |
         | 16 |  225 |
         | 17 |  256 |
         | 18 |  289 |
         | 19 |  324 |
         |    |  361 |
         | 20 |  400 |
         +---+---+
only showing top 20 rows
scala> dfobj.show(40)
```

output:-
```
+---+----+
| _1 | _2 |
+---+----+
| 1 | 1 |
| 2 | 4 |
| 3 | 9 |
| 4 | 16 |
| 5 | 25 |
| 6 | 36 |
| 7 | 49 |
| 8 | 64 |
| 9 | 81 |
| 10 | 100 |
| 11 | 121 |
| 12 | 144 |
| 13 | 169 |
| 14 | 196 |
| 15 | 255 |
| 16 | 256 |
| 17 | 289 |
| 18 | 324 |
| 19 | 361 |
| 20 | 400 |
| 21 | 441 |
| 22 | 484 |
| 23 | 529 |
| 24 | 576 |
| 25 | 625 |
| 26 | 676 |
| 27 | 729 |
| 28 | 784 |
| 29 | 841 |
| 30 | 900 |
| 31 | 961 |
| 32 | 1024 |
| 33 | 1089 |
| 34 | 1156 |
| 35 | 1225 |
| 36 | 1296 |
| 37 | 1369 |
| 38 | 1444 |
| 39 | 1521 |
| 40 | 1600 |
+---+----+
```

=> If we use only show it will display only 20 records in given table, if we having more than 20 records, we should mention the number in show() → it is an action.



Eg: The number should be above 10 & the square should be less than 1500

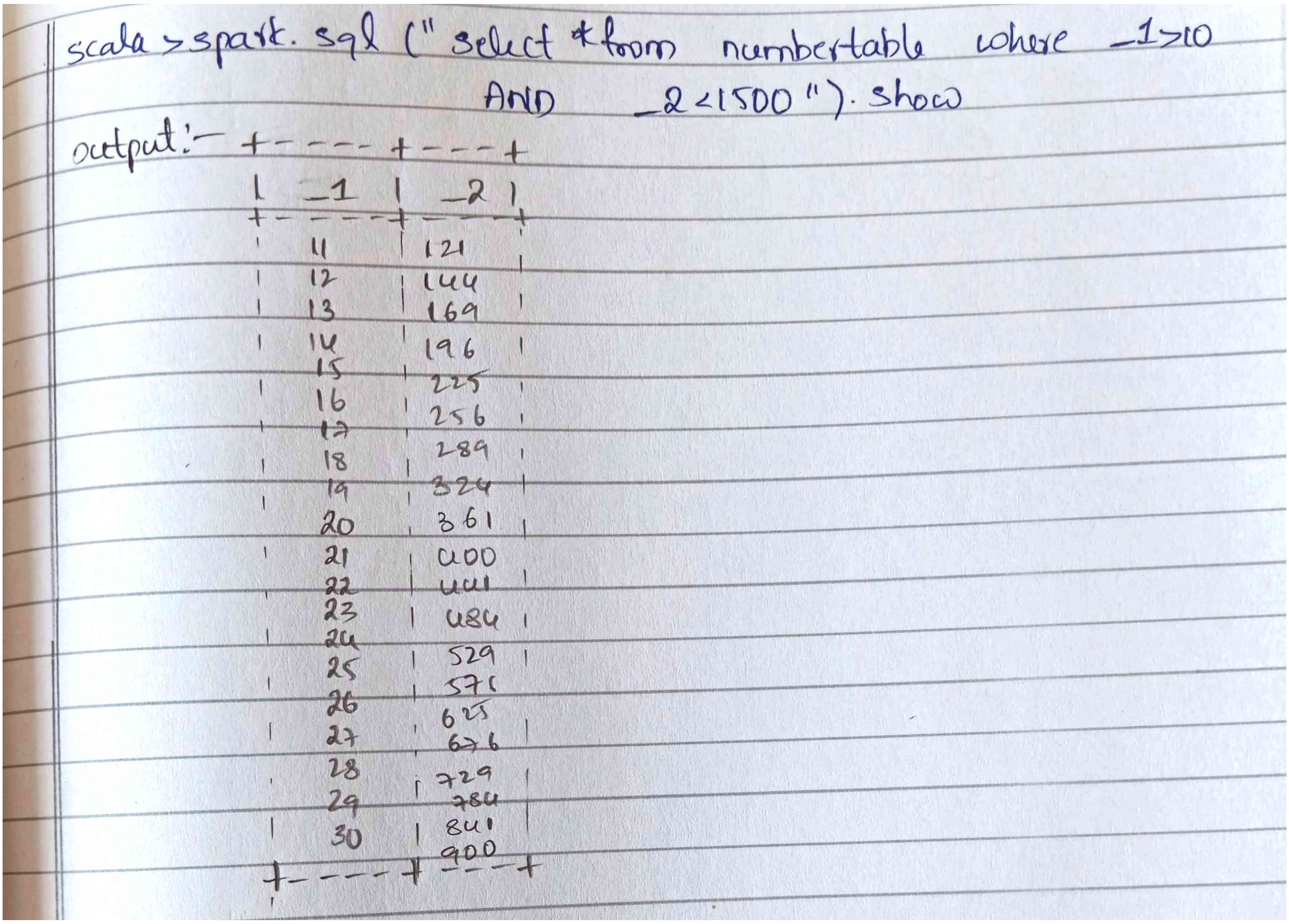
```
scala > spark.sql ("show tables"). show
scala > dfobj.registerTemptable ("numbertable")
scala > spark.sql ("show tables"). show
```

(without table name)

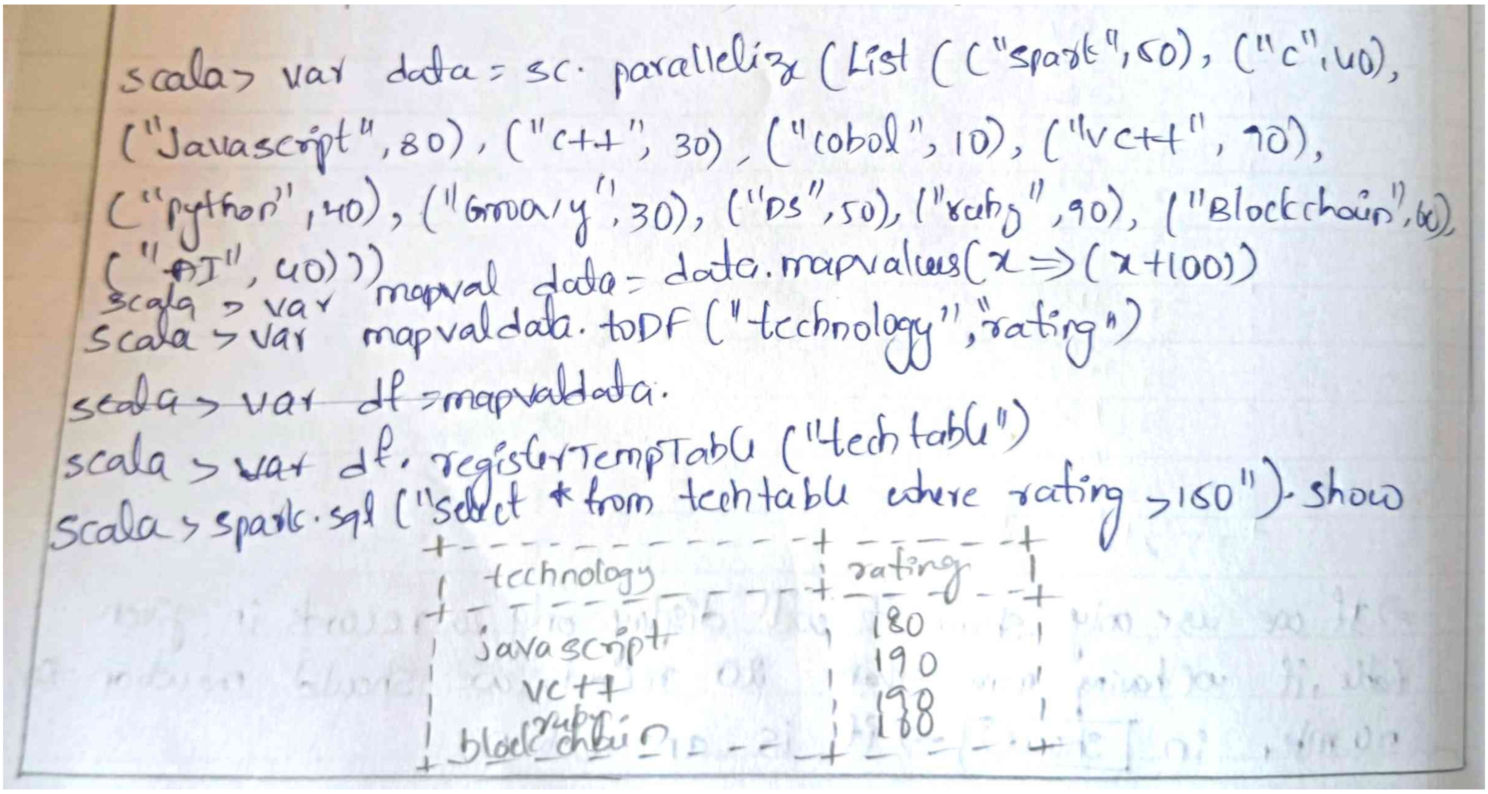
```
output:-
+---------+-----------+-------------+
|namespace|tableName  |isTemporary  |
+---------+-----------+-------------+
|         |           |
+---------+-----------+
```

(with table name)

```
+---------+-------------+-------------+
|namespace| tableName   | isTemporary |
+---------+-------------+-------------+
|         | numbertable |    true     |
+---------+-------------+-------------+
```



```
scala>spark.sql ("select *from numbertable where _1>10
                  AND   _2<1500").show
output:- +----+----+
         | _1 | _2 |
         +----+----+
         | 11 | 121|
         | 12 | 144|
         | 13 | 169|
         | 14 | 196|
         | 15 | 225|
         | 16 | 256|
         | 17 | 289|
         | 18 | 289|
         | 19 | 324|
         | 20 | 361|
         | 21 | 400|
         | 22 | 441|
         | 23 | 484|
         | 24 | 529|
         | 25 | 576|
         | 26 | 625|
         | 27 | 676|
         | 28 | 729|
         | 29 | 784|
         | 30 | 841|
         |    | 900|
         +----+----+
```



```
scala> var data = sc. paralleliz (List ( ("spark", 50), ("c", 40),
("Javascript", 80), ("c++", 30), ("cobol", 10), ("vc++", 90),
("python", 40), ("Groovy", 30), ("DS", 50), ("ruby", 90), ("Blockchain", 60),
("AI", 40)))
scala > var  mapval data = data.mapvalues(x => (x+100))
scala > var  mapvaldata. toDF ("technology", "rating")
~~scala > var  df =mapvaldata.~~
scala > var df. registerTempTable ("techtable")
scala > spark.sql ("select * from techtable where rating >150").show
        +--------------------+----------+
        | technology         | rating   |
        +--------------------+----------+
        | javascript         | 180      |
        | vc++               | 190      |
        |   ruby             | 198      |
        | blockchain         | 188      |
        +--------------------+----------+
```