**Quiz**

**Question 1: What functions must a dataset implement in order to be an RDD?**

partitions, iterator and dependencies

Correct: The RDD must have these three functions in order to be named 'resilient' and 'distributed' dataset.

**Question 2: Mark all the things that can be used as RDDs.**

HDFS file

Correct: This is the example from the video 'RDDs'.

A set of CSV files in my home folder

Correct: You can treat every file as a partition, why not?

MySQL table

Correct: You can partition the table by its primary key and use it as the data source.

In-memory array

Correct: This is the example from the video 'RDDs'.

**Question 3: Is it possible to access a MySQL database from within the predicate in the 'filter' transformation?**

Yes, but one need to create a database handle within the closure and close it upon returning from the predicate.

Correct: However, that is not an efficient solution. A better way would be to use the 'mapPartition' transformation which would allow you to reuse the handle between the predicate calls.

**Question 4: True or false? Mark only the correct statements about the 'filter' transform.**

There is the same number of partitions in the transformed RDD as in the source RDD.

Correct: Filtering establishes one-to-one correspondence between the partitions.

There is a single dependency on an input partition for every output partition.

Correct: Filtering establishes narrow dependencies between RDDs.

**Question 5: True or false? Mark only the incorrect statements.**

There is no native join transformation in Spark.

Correct: Incorrect statement. Rewatch the video 'Transformations 2'.

You cannot do a map-side join or a reduce-side join in Spark.

Correct: Incorrect statement. Every MapReduce computation could be expressed in Spark terms. Therefore, map-side joins and reduce-side joins could be expressed in Spark as well. But nobody does this in practice.

Spark natively supports only inner joins.

Correct: Incorrect statement. There are outer joins as well.

There is a native join transformation in Spark, and its type signature is: RDD , RDD => RDD .

Correct: Incorrect statement. Join keys must be explicit in the RDD items.

**Question 6: Mark all the transformations with wide dependencies. Try to do this without sneaking into the documentation.**

reduceByKey

Correct: Reduction requires data shuffling to regroup data items -- thus it has wide dependencies.

distinct

Correct: This is a kind of reduce-style operation, which requires a shuffle.

repartition

Correct: Repartitioning may join or split partitions.

join

Correct: This transformation requires a data shuffle -- this it has wide dependencies.

cartesian

Correct: Cartesian product is a kind of all-to-all join, it has wide dependencies.

**Question 7: Imagine you would like to print your dataset on the display. Which code is correct (in Python)?**

myRDD.collect().map(print)

Correct: You need to collect data to the driver program first.

**Question 8: Imagine you would like to count items in your dataset. Which code is correct (in Python)?**

def sum\_func(a, x):

a += 1

return a

myRDD.fold(0, sum\_func)

Correct: The 'fold' transformation updates an accumulator (which is zero initially) by calling the given function (which increments the value).

**Question 9: Consider the following implementation of the 'sample' transformation:**

**class MyRDD(RDD):**

**def my\_super\_sample(self, ratio):**

**return this.filter(lambda x: random.random() < ratio)**

**Are there any issues with the implementation?**

Yes, it exhibits nondeterminism thus making the result non-reproducible.

Correct: The major issue here is the random number generation. Two different runs over a dataset would lead to two different outcomes.

**Question 10: Consider the following action that updates a counter in a MySQL database:**

**def update\_mysql\_counter():**

**handle = connect\_to\_mysql()**

**handle.execute("UPDATE counter SET value = value + 1")**

**handle.close()**

**myRDD.foreach(update\_mysql\_counter)**