Name:

Date:

1. Given the below code. Choose the appropriate output?

from pyspark.sql import Row

from pyspark.sql import functions

from pyspark.sql import Column

import pyspark.sql.functions as func

a = [1002, 3001, 4002, 2003, 2002, 3004, 1003, 4006]

c=sc.parallelize(a).map(lambda x:Row(value=x))

b = spark.createDataFrame(c)

df=b.withColumn("x", (b.value /1000).cast("int"))

s=df.groupBy(df.x).sum("value")

s.select(s.x,func.col("sum(value)").alias("sum")).show()

a.

+-----+----+

| x| sum|

+-----+----+

|4.002|4002|

|3.004|3004|

|1.003|1003|

|2.002|2002|

|1.002|1002|

|4.006|4006|

|2.003|2003|

|3.001|3001|

+-----+----+

**b.**

**+---+----+**

**| x| sum|**

**+---+----+**

**| 1|2005|**

**| 3|6005|**

**| 4|8008|**

**| 2|4005|**

**+---+----+**

c.

+---+----------+

| x|sum(value)|

+---+----------+

| 1| 2005|

| 3| 6005|

| 4| 8008|

| 2| 4005|

+---+----------+

d.

+-----+----------+

| x|sum(value)|

+-----+----------+

|4.002| 4002|

|3.004| 3004|

|1.003| 1003|

|2.002| 2002|

|1.002| 1002|

|4.006| 4006|

|2.003| 2003|

|3.001| 3001|

+-----+----------+

2. Select the code snippet which find the word count for the words starting with ‘S’

a.

rddSc=sc.textFile("/FileStore/tables/wc.txt")

newRDD=rddSc.flatMap(lambda x:x.split(" ")).filter(lambda x:x.startswith('S')).reduceByKey(\_+\_)

newRDD.collect()

b.

rddSc=sc.textFile("/FileStore/tables/wc.txt")

newRDD=rddSc.map(lambda x:x.split(" ")).filter(lambda x:x.startswith('S'))

newRDD.map(lambda x:(x,1)).reduceByKey(lambda x,y:x+y).collect()

c.

rddSc=sc.textFile("/FileStore/tables/wc.txt")

newRDD=rddSc.flatMap(lambda x:x.split(" ")).filter(lambda x:x.startswith('S'))

newRDD.map(lambda x:(x,2)).reduceByKey(lambda x,y:x+y).collect()

**d.**

**rddSc=sc.textFile("/FileStore/tables/wc.txt")**

**newRDD=rddSc.flatMap(lambda x:x.split(" ")).filter(lambda x:x.startswith('S'))**

**newRDD.map(lambda x:(x,1)).reduceByKey(lambda x,y:x+y).collect()**

3. Which of the below methods act a action on Dataframe[select 3]

**collect**

**drop**

**limit**

**printSchema**

**registerTempTable**

**sample**

**first**

**count**

**distinct**

4. Which of the below methods act as transformation on Dataframe[select 3]

**collect**

**drop**

**limit**

**printSchema**

**registerTempTable**

**sample**

**first**

**explain**

5. In the given below code, match the appropriate set

df12=spark.range(1, 20)

df12.createOrReplaceTempView("test")

def squared(s):

return s \* s

k=spark.udf.register("squaredWithPython", <GREEN>)

display(df12.select("id", <RED>("id").alias("id\_squared")))

spark.sql("select id, <BLUE>(id) as id\_squared from test").show()



GREEN -k

BLUE -squaredWithPython

RED - squared

b.

GREEN -squared

BLUE - k

RED - squaredWithPython

c.

GREEN - squaredWithPython

BLUE -k

RED – squared

**d.**

**GREEN -squared**

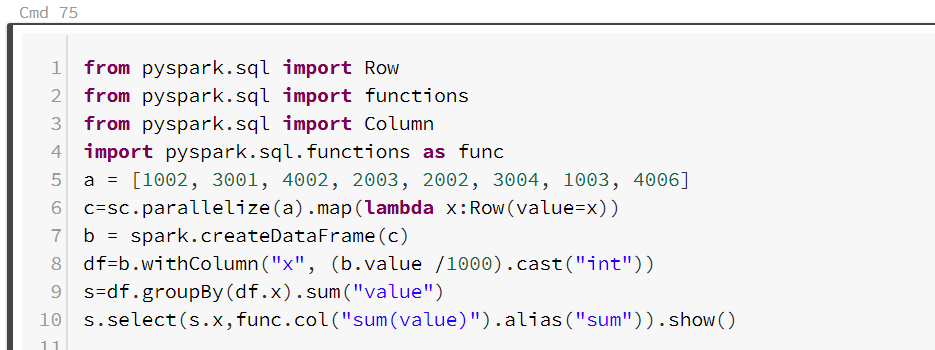
**BLUE -squaredWithPython**

**RED – k**

6. Which of the following statement is true

1. By default cache will make RDD to MEMORY and DataFrame to DISK
2. By default cache will make both RDD and DataFrame to MEMORY
3. By default cache will make RDD to MEMORY and DataFrame to MEMORY\_AND\_DISK
4. By default cache will make RDD to MEMORY\_AND\_DISK and DataFrame to MEMORY

7. Which line of the below code initiates shuffle/wide transformation,



1. LINE 8
2. LINE 7
3. LINE 9
4. LINE 10

8. We have dataframe df with below values.

+----+-------+

| age| name|

+----+-------+

|null|Michael|

| 30| Andy|

| 19| Justin|

+----+-------+

Select appropriate output produced by below code,

from pyspark.sql import Row

from pyspark.sql import functions

from pyspark.sql import Column

import pyspark.sql.functions as func

g=df.withColumn("nn",df.name.substr(1,1))

b=g.na.fill(100)

c=b.drop('name')

c.orderBy(c.age).limit(1).show()

a.

+---+---+

|age| nn|

+---+---+

| 19| J|

+---+---+

b.

+---+---+

|age| nn|

+---+---+

| 100| M|

+---+---+

c.

+----+-------+---+

| age| name| nn|

+----+-------+---+

|null|Michael| M|

+----+-------+---+

d.

+----+---+

| age| nn|

+----+---+

|null| M|

+----+---+

1. Which of the following DataFrame operations are wide transformations (that is, they result in a shuffle)(choose 3)?

A. repartition()  
B. filter()  
C. orderBy()  
D. distinct()  
E. drop()  
F. cache()

10. Given an instance of SparkSession named spark, review the following code:

import org.apache.spark.sql.functions.\_

a = Array(1002, 3001, 4002, 2003, 2002, 3004, 1003, 4006)

2,1,2,3,2,4,3,6

b = spark

.createDataset(a)

.withColumn("x", col("value") % 1000)

val c = b

.groupBy(col("x"))

.agg(count("x"), sum("value"))

.drop("x")

.toDF("count", "total")

.orderBy(col("count").desc, col("total"))

.limit(1)

.show()

Which of the following results is correct?

A.

+-----+-----+

|count|total|

+-----+-----+

| 3| 7500|

+-----+-----+

B.

+-----+-----+

|count|total|

+-----+-----+

| 1| 3001|

+-----+-----+

C.

+-----+-----+

|count|total|

+-----+-----+

| 3| 7006|

+-----+-----+

D.

+-----+-----+

|count|total|

+-----+-----+

| 8|20023|

+-----+-----+

11. Consider the following DataFrame:

val rawData = Seq(

(1, 1000, "Apple", 0.76),

(2, 1000, "Apple", 0.11),

(1, 2000, "Orange", 0.98),

(1, 3000, "Banana", 0.24),

(2, 3000, "Banana", 0.99)

)

val dfA = spark.createDataFrame(rawData).toDF("UserKey", "ItemKey", "ItemName", "Score")

Select the code fragment that produces the following result:

+-------+-----------------------------------------------------------------+

|UserKey|Collection |

+-------+-----------------------------------------------------------------+

|1 |[[0.98, 2000, Orange], [0.76, 1000, Apple], [0.24, 3000, Banana]]|

|2 |[[0.99, 3000, Banana], [0.11, 1000, Apple]] |

+-------+-----------------------------------------------------------------+

A.

import org.apache.spark.sql.expressions.Window

dfA.withColumn(

"Collection",

collect\_list(struct("Score", "ItemKey", "ItemName")).over(Window.partitionBy("ItemKey"))

)

.select("UserKey", "Collection")

.show(20, false)

B.

dfA.groupBy("UserKey")

.agg(collect\_list(struct("Score", "ItemKey", "ItemName")))

.toDF("UserKey", "Collection")

.show(20, false)

C.

dfA.groupBy("UserKey", "ItemKey", "ItemName")

.agg(sort\_array(collect\_list(struct("Score", "ItemKey", "ItemName")), false))

.drop("ItemKey", "ItemName")

.toDF("UserKey", "Collection")

.show(20, false)

D.

dfA.groupBy("UserKey")

.agg(sort\_array(collect\_list(struct("Score", "ItemKey", "ItemName")), false))

.toDF("UserKey", "Collection")

.show(20, false)

12. Consider the following DataFrame:

import org.apache.spark.sql.functions.\_

val people = Seq(

("Ali", 0, Seq(100)),

("Barbara", 1, Seq(300, 250, 100)),

("Cesar", 1, Seq(350, 100)),

("Dongmei", 1, Seq(400, 100)),

("Eli", 2, Seq(250)),

("Florita", 2, Seq(500, 300, 100)),

("Gatimu", 3, Seq(300, 100))

)

.toDF("name", "department", "score")

Select the code fragment that produces the following result:

+----------+-------+-------+

|department| name|highest|

+----------+-------+-------+

| 0| Ali| 100|

| 1|Dongmei| 400|

| 2|Florita| 500|

| 3| Gatimu| 300|

+----------+-------+-------+

A.

val maxByDept = people

.withColumn("score", explode(col("score")))

.groupBy("department")

.max("score")

.withColumnRenamed("max(score)", "highest")

maxByDept

.join(people, "department")

.select("department", "name", "highest")

.orderBy("department")

.dropDuplicates("department")

.show()

B.

people

.withColumn("score", explode(col("score")))

.orderBy("department", "score")

.select(col("name"), col("department"), first(col("score")).as("highest"))

.show()

C.

import org.apache.spark.sql.expressions.Window

val windowSpec = Window.partitionBy("department").orderBy(col("score").desc)

people

.withColumn("score", explode(col("score")))

.select(

col("department"),

col("name"),

dense\_rank().over(windowSpec).alias("rank"),

max(col("score")).over(windowSpec).alias("highest")

)

.where(col("rank") === 1)

.drop("rank")

.orderBy("department")

.show()

D.

people

.withColumn("score", explode(col("score")))

.groupBy("department")

.max("score")

.withColumnRenamed("max(score)", "highest")

.orderBy("department")

.show()

13. With respect to spark streaming widow operation for batch interval 10 seconds. Which of the below is not true statement?

a. windowedWordCounts = pairs.reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30 seconds, 10 seconds)

b. windowedWordCounts = pairs.reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30 seconds, 10 minutes)

c. windowedWordCounts = pairs.reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30 minutes, 10 minutes)

d. windowedWordCounts = pairs.reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30 minutes, 10 seconds)

14. Which of the following standard Structured Streaming sink types are idempotent and can provide end-to-end exactly-once semantics in a Structured Streaming job?

A. Console

B. Kafka

C. File

D. Memory

15. Which of following statements regarding caching are TRUE(choose 2)?

A. The default storage level for a DataFrame is StorageLevel.MEMORY\_AND\_DISK.

B. The uncache() method evicts a DataFrame from cache.

C. The persist() method immediately loads data from its source to materialize the DataFrame in cache.

D. Explicit caching can decrease application performance by interfering with the Catalyst optimizer's ability to optimize some queries.

16. What is the output of the code?

df1=spark.range(1, 10)

df1.filter(df1.id%2==0).orderBy(df1.id.desc()).limit(1).show()

a.

+---+

| id|

+---+

| 2||

+---+

b.

+---+

| id|

+---+

| 8|

+---+

c.

+---+

| id|

+---+

| 10|

+---+

d.

+---+

| id|

+---+

| 9|

+---+