**Instructions**. Answer the following multiple choice questions by selecting the correct choices.

1. **Principles of (functional) programming**

   (a) Which of the following are programming paradigms? (Select three.)

   √ ***Declarative*** √ ***Functional*** □ Hadoop √ ***Imperative*** □ Scala

   (b) What three concepts characterize a purely functional programming language?

   √ ***referential transparency*** □ input/output √ ***no side effects*** □ procedural √ ***immutability***

2. **Big data properties**.

   (a) In lecture we discussed the meaning of the term "Big Data." We decided that, for simplicity, we will call data "big" when it is

   □ at least 1Gb

   √ ***too big to fit in fast memory (cpu cache + ram) on a single compute node***

   □ too big to fit in all computer memory (whether fast or slow)

   □ too big to be dealt with by traditional data-processing software

   Explanation. We agreed that a good definition of "Big Data" for our purposes is data that is too large to fit in the fast computer memory of a single machine. Although Wikipedia has an alternative definition—data that is "too big to be dealt with by traditional data-processing application software"—and while that definition is not wrong, it is not the definition we agreed upon in this class.

   (b) "Big Data" concerns which of the following types of data?

   □ structured □ semi-structured □ unstructured √ ***all of these***

   Explanation. Big Data is a blanket term for the data that are too large in size and complex in nature, and which may be structured, unstructured, or semi-structured, and may also be arriving at high velocity.

   (c) JSON and XML are examples of which type of data?

   □ structured □ unstructured √ ***semi-structured*** □ none of these

   Explanation. Semi-structured data are that which have a structure but do not fit into the relational database. Semi-structured data are organized, which makes it easier for analysis when compared to unstructured data. JSON and XML are examples of semi-structured data.

   (d) Which two of the following statements are true of unstructured data?

   □ It is generally easier to analyze than other types of data.

   √ ***It is often referred to as "messy" data.***

   □ It fits neatly into a schema.

   √ ***It is the most widespread type of data.***

   □ It is usually found in tables.

3. **Latency and fault-tolerance**.

(a) *Latency* is degradation in performance due to...

□ a small number of cores in the central processing unit

√ ***slow data transfer across the network or cluster***

√ ***shuffling data between different nodes in a cluster***

□ failure of one or more nodes in the cluster

□ stack overflow caused by recursion

(b) Hadoop achieves fault-tolerance by...

□ using lazy evaluation and garbage collection.

√ ***writing intermediate computations to disk.***

□ keeping all data immutable and in-memory.

□ replaying functional transformations over the original (immutable) dataset.

(c) Spark decreases latency while remaining fault-tolerant by...

□ using ideas from imperative programming.

√ ***using ideas from functional programming.***

□ discarding data when it's no longer needed.

√ ***keeping all data immutable and in-memory.***

√ ***replaying functional transformations over the original (immutable) dataset.***

4. **Transformations and actions**.

(a) In Spark a **transformation** on an RDD...

□ is eagerly evaluated.<br>
√ ***is lazily evaluated.***<br>
□ immediately computes and returns a result.<br>
√ ***does not immediately compute a result.***<br>
√ ***usually returns another RDD (once it's evaluated).***

(b) In Spark an **action** on an RDD...

√ ***is eagerly evaluated.***<br>
□ is lazily evaluated.<br>
√ ***immediately computes and returns a result.***<br>
□ does not immediately compute a result.<br>
□ always returns another RDD (once it's evaluated).

(c) After performing a series of transformations on an `RDD`, which of the following methods could you use to make sure those transformations are not repeated (e.g., on each iteration of an algorithm)?

□ `save`<br>
√ *`persist`*<br>
□ `memoize`<br>
□ There is no such method because of the JVM's garbage collection mechanism.

(d) Why does Spark's `RDD` class not have a `foldLeft` method?

□ `foldLeft` can only be performed on lists of Boolean values.<br>
□ `foldLeft` doesn't work on immutable collections.<br>
□ `foldLeft` is not stack-safe.<br>
□ `foldLeft` is not fault-tolerant.<br>
√ ***`foldLeft` is not parallelizable.***

(e) Why is available in Spark's `RDD` class that overcomes the limitation of `foldLeft` mentioned in the previous part of this exercise?

√ *`aggregate`* □ `fold` □ `foldLeft` □ `join` □ `leftOuterJoin`

5. **Read the docs**. Navigate to the Spark API documentation at
   `https://spark.apache.org/docs/3.3.1/api/scala/org/apache/spark/index.html`
   Enter "RDD" in the search box and select `RDD` from the results that appear on the left.

   (a) Scroll down the resulting `RDD` API documentation page and find the `cache()` method. What does it say?

   √ ***Persist this RDD with the default storage level (*`MEMORY_ONLY`*).***

   □ Mark the RDD as non-persistent, and remove all blocks for it from memory and disk.

   □ Set this RDD's storage level to persist its values across operations after the first time it is computed.

   □ Save this RDD as a SequenceFile of serialized objects.

   (b) On the `RDD` API doc page, find the version of `persist` that takes an argument: `def persist(newLevel: StorageLevel)`. What does it say?

   □ Persist this RDD with the default storage level (`MEMORY_ONLY`).

   □ Mark the RDD as non-persistent, and remove all blocks for it from memory and disk.

   √ ***Set this RDD's storage level to persist its values across operations after the first time it is computed.***

   □ Save this RDD as a SequenceFile of serialized objects.

   (c) On the `RDD` API doc page, find the `unpersist` method. What does it say?

   □ Persist this RDD with the default storage level (`MEMORY_ONLY`).

   √ ***Mark the RDD as non-persistent, and remove all blocks for it from memory and disk.***

   □ Set this RDD's storage level to persist its values across operations after the first time it is computed.

   □ Save this RDD as a SequenceFile of serialized objects.

   (d) What's the difference between the `sample` and `takeSample` methods of the `RDD` class?

   □ `sample` always uses a with-replacement sampling method, while `takeSample` always samples without replacement.

   √ ***`sample` returns an RDD, while `takeSample` returns an Array.***

   √ ***The second argument specifies the number of samples desired either as a fraction of the size of the RDD (`sample`) or as an absolute number (`takeSample`).***

   □ There is no difference; they are just two different names one can use to invoke the same function.