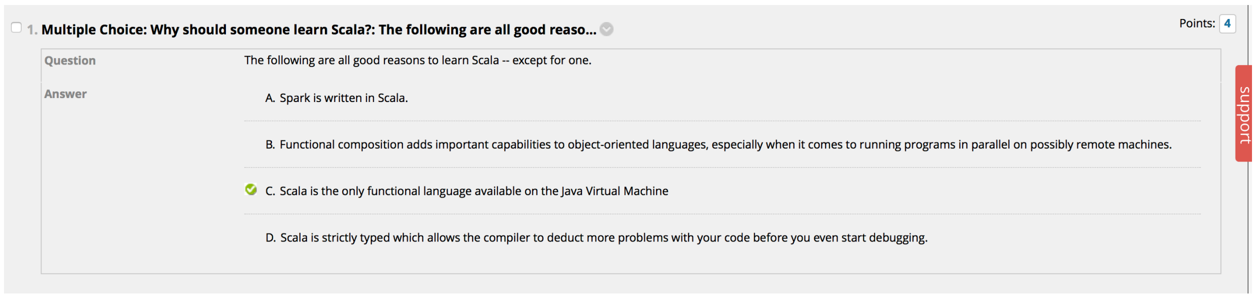
Final Exam CSYE 7200 Spring 2018


1. Multiple Choice: Why should someone learn Scala?: The following are all good reaso...

Points: 4

Question: The following are all good reasons to learn Scala -- except for one.

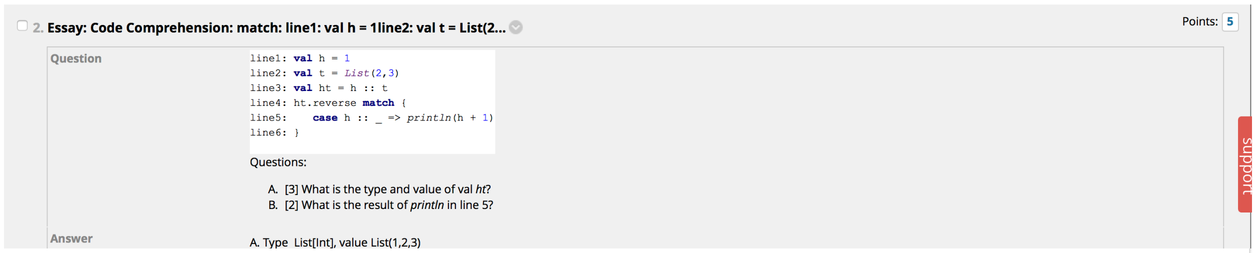
Answer:

A. Spark is written in Scala.

B. Functional composition adds important capabilities to object-oriented languages, especially when it comes to running programs in parallel on possibly remote machines.

✓ C. Scala is the only functional language available on the Java Virtual Machine

D. Scala is strictly typed which allows the compiler to deduct more problems with your code before you even start debugging.


2. Essay: Code Comprehension: match: line1: val h = 1line2: val t = List(2...

Points: 5

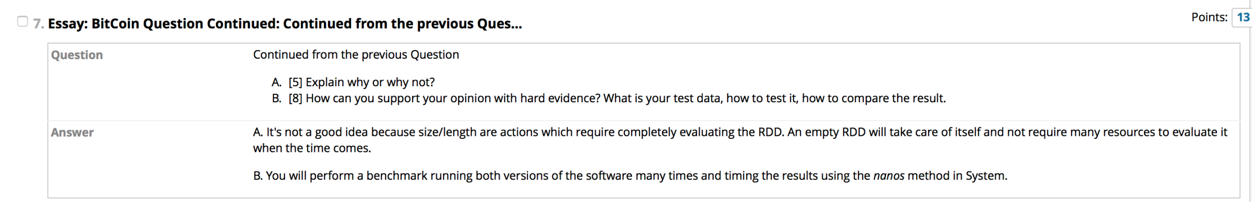
Question:

```
line1: val h = 1
line2: val t = List(2,3)
line3: val ht = h :: t
line4: ht.reverse match {
line5:     case h :: _ => println(h + 1)
line6: }
```

Questions:

A. [3] What is the type and value of val *ht*?
B. [2] What is the result of *println* in line 5?

Answer: A. Type List[Int], value List(1,2,3)


7. Essay: BitCoin Question Continued: Continued from the previous Ques...

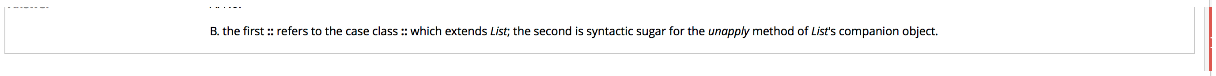
Points: 13

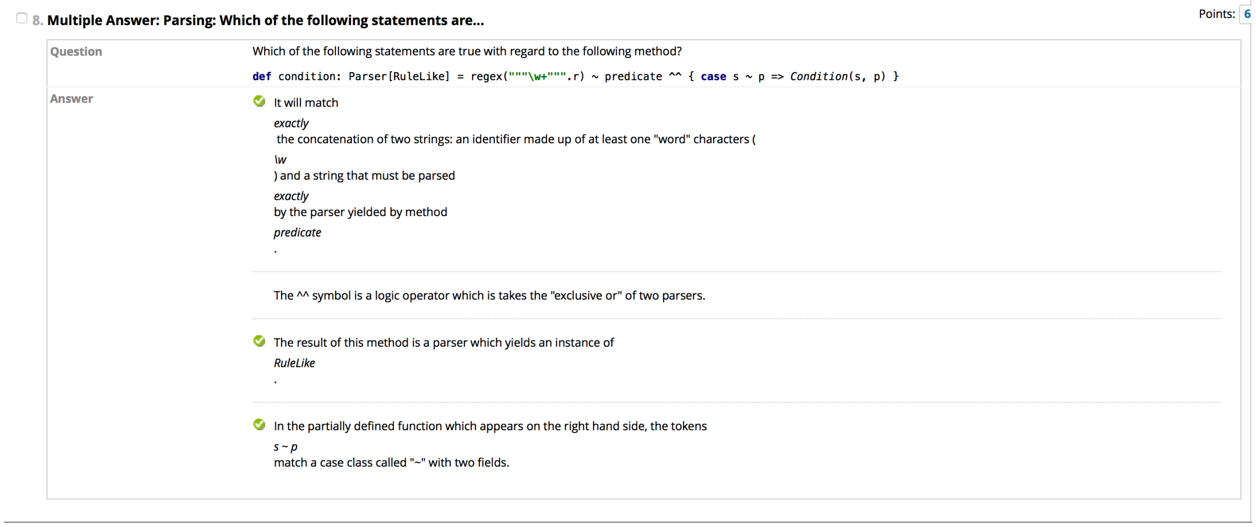
Question: Continued from the previous Question

A. [5] Explain why or why not?
B. [8] How can you support your opinion with hard evidence? What is your test data, how to test it, how to compare the result.

Answer: A. It's not a good idea because size/length are actions which require completely evaluating the RDD. An empty RDD will take care of itself and not require many resources to evaluate it when the time comes.

B. You will perform a benchmark running both versions of the software many times and timing the results using the *nanos* method in System.


B. the first :: refers to the case class :: which extends *List*; the second is syntactic sugar for the *unapply* method of *List*'s companion object.


8. Multiple Answer: Parsing: Which of the following statements are...

Points: 6

Question: Which of the following statements are true with regard to the following method?

```
def condition: Parser[RuleLike] = regex("""\w+""".r) ~ predicate ^^ { case s ~ p => Condition(s, p) }
```

Answer:

✓ It will match *exactly* the concatenation of two strings: an identifier made up of at least one "word" characters (\w) and a string that must be parsed *exactly* by the parser yielded by method *predicate* .

The ^^ symbol is a logic operator which is takes the "exclusive or" of two parsers.

✓ The result of this method is a parser which yields an instance of *RuleLike* .

✓ In the partially defined function which appears on the right hand side, the tokens *s ~ p* match a case class called "~" with two fields.

9. Multiple Answer: Scala code comprehension and enhancement: Please ensure that you have the lates...

Points: 4

Question

Please ensure that you have the latest code from Majabigwaduce (by doing a pull). Open the project in your IDE, although you should be able to answers these questions without using your IDE, using it might be easier. If you have any difficulties, you can simply look at the code using your browser on the repository.

Majabigwaduce is a relatively simple framework for doing remote parallel computations using Akka actors. It follows the Map-Reduce paradigm of other systems, providing a low-level applications programming interface based on Mappers and Reducers similar to Hadoop's Java API. The *CountWords* example uses that API.

There is, in addition, a higher-level API which is similar to the way Spark works, in that the programmer creates a lazy, partitionable dataset and manipulates it using transformations and actions, similar to Spark's *RDD*. The in this case is called *DataDefinition.*

It has three important methods:

- `def apply(): Future[Map[K, V]]`
- `def map[L, W: Monoid](f: ((K,V)) => (L,W)): DataDefinition[L, W]`
- `def aggregate[W: Zero](f: (W, V) => W): Future[W]`

The *map* method simply returns another *DataDefinition*. The other two return zero- or one-dimensional collections.

Please ensure that you have looked at all parts of this question. Make sure you label each of your answers with the correct part letter. As always, you are to provide brief answers that match what I'm looking for. No extra credit for writing essays!!

Question:

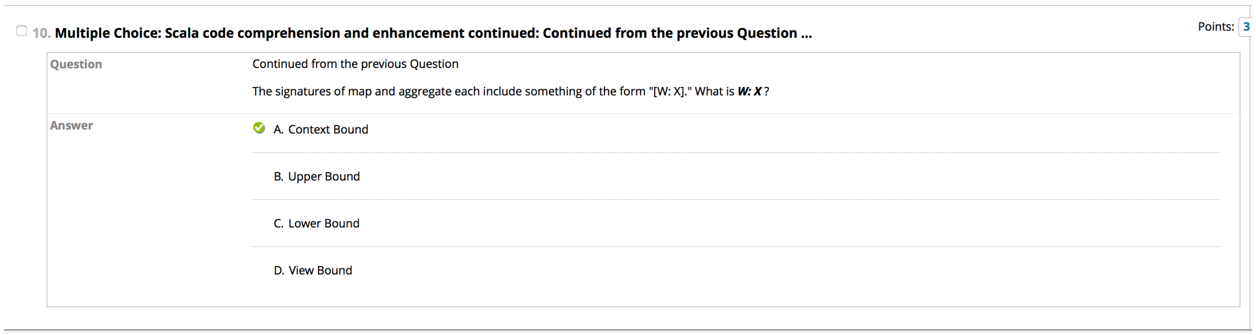
Which of the three methods can be considered a "transformation" (as opposed to an "action")?

Answer

A. apply

[✓ B. map]

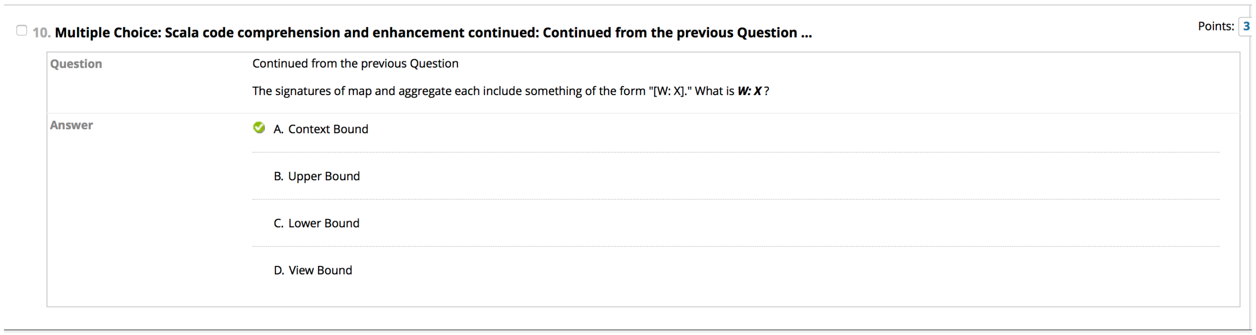
C. aggregate



☐ 10. **Multiple Choice: Scala code comprehension and enhancement continued: Continued from the previous Question ...**

Points: 3

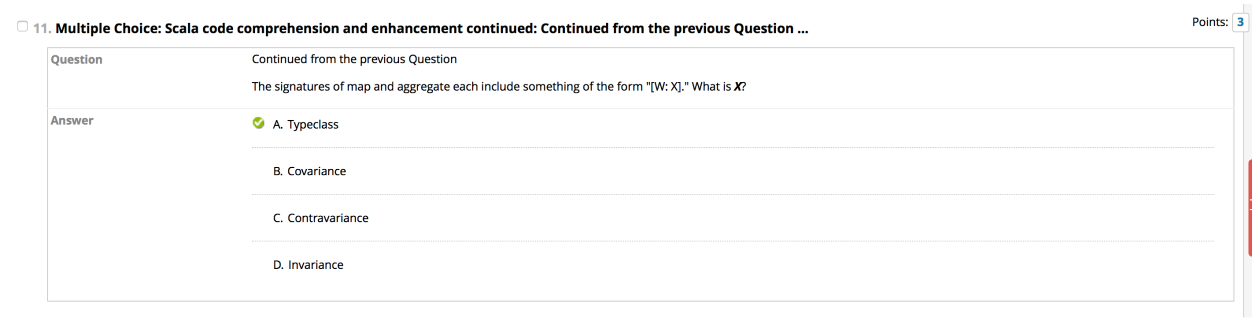
Question	Continued from the previous Question The signatures of map and aggregate each include something of the form "[W: X]." What is ***W: X*** ?
Answer	✓ A. Context Bound B. Upper Bound C. Lower Bound D. View Bound



☐ 10. **Multiple Choice: Scala code comprehension and enhancement continued: Continued from the previous Question ...**

Points: 3

Question	Continued from the previous Question The signatures of map and aggregate each include something of the form "[W: X]." What is ***W: X*** ?
Answer	✓ A. Context Bound B. Upper Bound C. Lower Bound D. View Bound



☐ 11. **Multiple Choice: Scala code comprehension and enhancement continued: Continued from the previous Question ...**

Points: 3

Question	Continued from the previous Question The signatures of map and aggregate each include something of the form "[W: X]." What is ***X***?
Answer	✓ A. Typeclass B. Covariance C. Contravariance D. Invariance

12. Short Answer: Scala code comprehension and enhancement continued: Continued from the previous Question ...

Points: 8

Question	Continued from the previous Question A. [3] Can you explain--briefly--what this "[W: X]" actually means in practical terms? or why the author uses it here? B. [5] Why do you think the author chose to use this mechanism rather than having *W* simply extend another type? This part is somewhat harder than the others.
Answer	A. It means that there is an implicit variable available of type *X[W]* that can be used to implement the methods of *X[W]*. A less elegant-looking form of this would be simply to pass in a parameter of type *X[W]*. B. It's used here because it is not convenient to redefine *W* to support the required methods possibly because *W* is a (third-party) library and/or final class.

13. Short Answer: Scala code comprehension and enhancement continued: Continued from the previous Question ...

Points: 8

Question	Continued from the previous Question There is a method with a "negative-one-dimensional" result missing: that would be *count*. It should have a signature like `def count: Future[Int]`. Implement this method for *LazyDD* in a manner similar to the implementation of *aggregate*.
Answer	`for (kVm <- apply()) yield kVm.size`

14. Short Answer: Scala code comprehension and enhancement continued: Continued from the previous Question ...

Points: 9

Question	Continued from the previous Question A. [4] In the definition of *case class LazyDD*, there is a third parameter set marked *implicit*. This value (*context*) is not passed in explicitly by any invocation of *LazyDD*'s apply method. So, how does it get satisfied? B. [5] Also, in the definition of *LazyDD*, there is a curious extra set of parentheses around *(K,V)*. Can you explain why those are necessary (hard).
Answer	A. When the compiler has to look for an implicit, it first looks in the current scope, then in the object of one of a relevant type. Here, inside object *DataDefinition*, an implicit context : *DDContext* is defined. B. Because *(K, V)* is of a *Tuple*.

15. Short Answer: Scala code comprehension and enhancement continued: Continued from the previous Question ...

Points: 5

Question	Continued from the previous Question Can you think of another important type of transformation that the author has omitted? Note that, as a transformation, it must yield a *DataDefinition*, but it is not something that could be implemented by invoking *map* with an appropriate value for *f*. One more clue: it is not *flatMap*.
Answer	*filter*

16. Short Answer: Scala code comprehension and enhancement continued: Continued from the previous Question ...

Points: 5 (Extra Credit)

Question	Continued from the previous Question [5 bonus points] Implement your transformation method for the previous question.
Answer	`def filter(p: ((L, W)) => Boolean): DataDefinition[L, W] = LazyDD[K, V, L, W](kVm.filter(f andThen p), f)(partitions)`