This Test has 28 attempts. For information on editing questions, click **More Help** below.

Test Canvas: Mid-term Fall 2019- Requires Respondus LockDown Browser

The Test Canvas lets you add, edit, and reorder questions, as well as review a test. More Help

Question Settings

You can edit, delete, or change the point values of test questions on this page. If necessary, test attempts will be regraded after you submit your changes.

Description	This is your mid-term exam.
Instructions	
Total Questions	7
Total Points	68
Number of Attempts	28

Select: All None | Select by Type: - Question Type -

Delete and Regrade | Points | Update and Regrade | Hide Question Details

1. **Multiple Answer: Fake news? Or real?: Which of the following represent real...** Points: 10

Question Which of the following represent real news stories about Scala?

Answer

- ✓ Scala can now be used in Browser code (it compiles to javascript), natively, and on the JVM.
- Scala implements most code more efficiently than Java so, if you have a performance-critical task to be implemented on the JVM, you should use Scala instead of Java.
- ✓ The rewrite of Scala for version 3.0 and beyond (currently being released experimentally as "Dotty") is scheduled for full release in early 2020, soon after the release of the 2.14 version.
- The concept of "lambdas" was invented by Martin Odersky, one cool dude, and was ultimately copied by Java in Java 8.
- ✓ Scala version 2.13 is out and one of the major differences is that *Stream* has been deprecated in favor of *LazyList*.
- Scala is the greatest language of all time and the ideal language for experimenting in Data Science research (as opposed to Data Engineering). This is because Scala is a compiled, strictly-typed language. For this reason, you should stop using Python for Data Science work. Trust me. I know.

Implicits have been found to be too confusing for programmers to use and will therefore be unavailable in Scala 3.0. Sad.

Crooked Martin's *Scala* is a dead language because Java 11 has caught up. Everything that you can do in Scala can be done in Java now so there's no point in Scala.

✓ Functional programming has been around since the 1950s but Scala is the first mainstream FP language which is also object-oriented.

✓ Scala actors were deprecated a few versions ago. Instead, we use Akka for all applications that need to be reactive, including streaming operations.

2. Multiple Answer: Parameters: The presence or absence of param...

Points: 6

Question	The presence or absence of parameters has great significance. Which of the following pairs of concepts (or constructs) differ *primarily* (or *at least partly)* according to their having/not having parameters?
Answer	✓ A. *class* vs. *object* ✓ B. *trait* vs. *abstract class* ✓ C. *lazy val* vs. *def* D. *function* vs. *method* E. *lazy list* vs. *list* F. *match expression* vs. *partial function* G. *call by name* vs. *call by value*

3. Multiple Answer: Call by name vs. function: Considering the declarations of metho...

Points: 6

Question	Considering the declarations of methods, which of the following are valid reasons why Scala has a special syntax for call-by-name parameters, for example: *x:* => *X*, as opposed to simply declaring the parameter as a function, for example x: () => X.
Answer	A. Because passing the function would force its evaluation. ✓ B. Because, when referencing the value x inside the method, you would need to apply the function by adding parentheses, for example *val y = x()* ✓ C. Because the caller of the method would typically have to pass a lambda of the form *() => x* as the argument to the parameter (instead of just passing *x*). D. Because passing a function would not be as efficient as the call-by-name mechanism.

4. Matching: Scala syntax: Match up the following left-hand-side...

Points: 9

Question Match up the following left-hand-sides of declarations with their right-hand-sides.

Answer

Match	Question Items	Answer Items
A. -	A. `val f: PartialFunction[Any, Int]`	A. `{ case x: Int => x; case x: String => x.toInt }`
B. -	B. `val a: AnyVal`	B. `if (true) 1`
C. -	C. `val (a, b)`	C. `"a" -> "b"`
D. -	D. `def method[T, R](f: T => T => R): T => T => R`	D. `a => b => f(b)(a)`
E. -	E. `def method[T, T, R](f: (T, T) => R): (Try[T], Try[T]) => Try[R]`	E. `map2(_,_)(f)`
F. -	F. `val a: Int`	F. `if (true) 1 else 0`

5. Multiple Answer: Variance: Which of the following statements reg...

Points: 10

Question Which of the following statements regarding variance are true?

Answer

✓ A.

Dealing with variance is necessary in a strictly-typed object-oriented language like Scala because of the implications of the Liskov Substitution Principle.

✓ B.

If container type *C* is covariant on *A*, i.e. *C* is defined something like **trait C[+A] {...}**, then *C[S]* is a sub-class of *C[T]* provided that *S* is a sub-class of *T*.

✓ C.

If container type *C* is contravariant on *A*, i.e. *C* is defined something like **trait C[-A] {...}**, then *C[T]* is a sub-class of *C[S]* provided that *S* is a sub-class of *T*.

D.

In Scala, an instance of *Array[S]* can be passed to a method that requires an *Array[T]* where *S* is a sub-class of *T*.

✓ E.

Covariant position refers to the result type of a method while *contravariant position* refers to a type referenced in a parameter of a method.

F. All mutable collections in Scala are declared to be covariant on the parametric (i.e. underlying) type.

6. Short Answer: Lazy Collection: You have been asked to

Points: 12

implement a La...

Question

You have been asked to implement a *LazyCollection*. Because the initial values of the collection are typically expensive to gather (requiring an internet lookup), we will store them as a call-by-name iterator. Whenever we invoke the *iterator* method on our *LazyCollection* it will force an evaluation of each element. Therefore we try to do that as few times as possible. Meanwhile, we can invoke *map* as often as we like because it is required to be very inexpensive. Here is the code you have been given--your task is to implement where you see the ???. Remember SOE.

```
case class LazyCollection[X, Y](private val f: X => Y)(xs: => Iterator[X]) {
  def map[Z](g: Y => Z): LazyCollection[X, Z] = ???
  def iterator: Iterator[Y] = xs map f
}

object LazyCollection extends App {
  val sequence = Seq(1,2,3)
  val lazyNumbers = LazyCollection[Int, Int](identity)(sequence.toIterator)
  val transformed = lazyNumbers map (_*2)
  transformed.iterator.foreach (println) // 2 4 6
}
```

Answer

```
LazyCollection(f andThen g)(xs)
```

7. Short Answer: Folding: A colleague has written a method to a...

Points: 15

Question

A colleague has written a method to apply an associative aggregation function to a sequence of Doubles. He insists that this cannot be accomplished without using a mutable variable. Please show him how it can be done without any mutable variables and without any danger of a stack overflow. [For full credit, you may not use any higher-order library functions--however, any answer that yields the correct result will get some credit].

His code:

```
object Aggregation {

  def aggregate[Double](xs: Seq[Double], f: (Double,Double)=>Double, initial: Double = 0): Double = {
    var result = initial
    xs foreach { x => result = f(result, x) }
    result
  }
}
```

Answer

```
object Aggregation {

  def aggregate(xs: Seq[Double], f: (Double, Double) => Double, initial: Double = 0): Double = {
    @scala.annotation.tailrec // not required for full marks
    def inner(r: Double, work: Seq[Double]): Double = work match {
      case Nil => r
      case h :: t => inner(f(r, h), t)
    }

    inner(initial, xs)
  }
}
```

Select: All None Select by Type: - Question Type -

Delete and Regrade | Points | Update and Regrade | Hide Question Details

← OK