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

1. **Scala Higher-order Functions**

   All parts of this question refer to the following `sum` function.

```
def sum(f: Int => Int): (Int, Int) => Int = {
  def sumF(a: Int, b: Int): Int = {
    if (a > b) 0
    else f(a) + sumF(a + 1, b)
  }
  sumF
}
```

(a) What does `sum(2, 3)` compute?

- ☐ `2 + 3`
- ☐ `2 + 2 + 3 + 3`
- ☐ `2 * 2 + 3 * 3`
- ☐ a function that takes two integer arguments and returns their sum
- ☐ `sum(2, 3)` causes a run-time error.
- ✓ ***`sum(2, 3)` causes a compile-time error.***

(b) What does `sum(x => x)(2, 3)` compute?

- ✓ *`2 + 3`*
- ☐ `2 + 2 + 3 + 3`
- ☐ `2 * 2 + 3 * 3`
- ☐ a function that takes two integer arguments and returns their sum
- ☐ `sum(x => x)(2, 3)` causes a run-time error.
- ☐ `sum(x => x)(2, 3)` causes a compile-time error.

(c) What does `sum(x => x)` return?

- ☐ `2 + 3`
- ☐ `2 + 2 + 3 + 3`
- ☐ `2 * 2 + 3 * 3`
- ✓ ***a function that takes two integer arguments and returns their sum***
- ☐ `sum(x => x)(2, 3)` causes a run-time error.
- ☐ `sum(x => x)(2, 3)` causes a compile-time error.

(d) What does `sum(x => x + x)(2, 3)` compute?

- ☐ `2 + 3`
- ✓ *`2 + 2 + 3 + 3`*
- ☐ `2 * 2 + 3 * 3`
- ☐ a function that takes two integer arguments and returns their sum
- ☐ `sum(x => x + x)(2, 3)` causes a run-time error.
- ☐ `sum(x => x + x)(2, 3)` causes a compile-time error.

(e) What does `sum(x => x * x)(2, 3)` compute?

- ☐ `2 + 3`
- ☐ `2 + 2 + 3 + 3`
- ✓ *`2 * 2 + 3 * 3`*
- ☐ a function that takes two integer arguments and returns their sum
- ☐ `sum(x => x * x)(2, 3)` causes a run-time error.
- ☐ `sum(x => x * x)(2, 3)` causes a compile-time error.

(f) What does `sum(x => x / 1.0)(2, 3)` compute?

- ☐ `2 + 3`
- ☐ `2 + 2 + 3 + 3`
- ☐ `2 * 2 + 3 * 3`
- ☐ a function that takes two integer arguments and returns their sum
- ☐ `sum(x => x/1.0)(2, 3)` causes a run-time error.
- ✓ ***`sum(x => x/1.0)(2, 3)` causes a compile-time error.***

2. **Scala Pattern Matching 1**

Consider the general form of pattern matching in Scala,

```
e match { case p1 => e1 ... case pn => en }
```

Which of the following are true statements?

- ☐ Scala matches the value of the selector `e` with the patterns `p1, ..., pn` in the order in which they are written.
- ☐ The match expression is rewritten to the right-hand side of the first case where the pattern matches the selector `e`.
- ☐ References to pattern variables are replaced by the corresponding parts in the selector.
- ✓ ***All of the above.***
- ☐ None of the above.

3. **Scala Pattern Matching 2**

Consider the following Scala program.

```
trait Expr
case class Number(n: Int) extends Expr
case class Sum(e1: Expr, e2: Expr) extends Expr

object Number{
  def apply(n: Int) = new Number(n)
}

object Sum{
  def apply(e1: Expr, e2: Expr) = new Sum(e1, e2)
}
```

```
def eval(e: Expr): Int = e match {
  case Number(n) => n
  case Sum(e1, e2) => eval(e1) + eval(e2)
}
```

What is the result of the following expression?

```
eval(Sum(Number(1), Number(2)))
```

☐ 0 ☐ 1 ☐ 2 √ *3* ☐ None of the above.

4. Consider the Scala code below.

```
val x = List(1,2,3)
val y = List(0, x, 4)
```

(a) What is the type of `x`?

☐ `List[T]`
√ *`List[Int]`*
☐ `List[Any]`
☐ `List[Nothing]`
☐ `List[Object]`

(b) What is the type of `y`?

☐ `List[T]`
☐ `List[Int]`
√ *`List[Any]`*
☐ `List[Nothing]`
☐ `List(0, x, 4)` causes a run-time error.
☐ `List(0, x, 4)` causes a compile-time error.

(c) What is `y.length`?

☐ 0
√ *3*
☐ 5
☐ `y.length` causes a run-time error.
☐ `y.length` causes a compile-time error.

(d) What is `x == List(1, 2, y.length)`?

☐ `List(1, 2, 3)`
☐ `List(x, 1, 2, 3)`
√ *`true`*
☐ `false`
☐ None of these.

5. Reducing lists with `foldLeft`.

   Suppose you want to implement a (polymorphic) `reverse` function, which reverses the order of a given list, `xs: List[T]`, using Scala's `foldLeft` function.

   You start with

```
def reverse[T](xs: List[T]): List[T] = (xs foldLeft ???)((ys, y) => ???)
```

   (a) What aspect of the code above tells you that this `reverse` function will be *polymorphic*?

   - ☐ It operates on lists.
   - ☐ The second `???` will be a function, so it's "higher-order."
   - ☐ There is a folding or "reduction" operation involved.
   - ☐ It is recursive.
   - √ ***It takes a type parameter*** ***`T`.***

   (b) The first set of three question marks `???` should be replaced with which of the following?

   - ☐ `Nil`
   - ☐ `List()`
   - √ *`List[T]()`*
   - ☐ `List[T](0)`
   - ☐ `ys :: y`
   - ☐ `y :: ys`

   (c) The second set of three question marks `???` should be replaced with which of the following?

   - ☐ `Nil`
   - ☐ `List()`
   - ☐ `List[T]()`
   - ☐ `List[T](0)`
   - ☐ `ys :: y`
   - √ *`y :: ys`*