Which of the following statements most precisely describes this function:

```scala
def f[A, B](xs: List[A])(g: A => B): List[B] = xs match {
  case Nil => Nil
  case x :: xs => g(x) :: f(xs)(g)
}
```

Select one answer

☐ It is a function that implements a fold

☐ It is a function that implements the Y combinator

☐ It is a function that implements a currying operation

☑ It is a function that implements a map

Which of the following statements most precisely describes Scala and its programming paradigm?

Select one answer

☐ Scala is a functional programming language

☐ Scala is an imperative programming language

☑ Scala is a programming language with both functional and imperative features.

The following program is implemented in the language with lazy evaluation from Week 6 of the lab assignments:

```
(letrec ((r (cons 0 s))
         (s (cons 1 r)))
  (force (head (tail (tail r)))))
```

What does it do?

Select one answer

☐ Returns a thunk

[✓ Returns the the number value 0]

☐ Returns the number value 1

☐ Either does not terminate or crashes with a stack overflow

Consider the following program:

```
(let ((f (lambda (g)
           (lambda (y) (g 10)))))
  ((f (lambda (x) (+ x y))) 3))
```

In a language *with dynamic scoping* what would be the outcome of running the program?

Select one answer

☐ 3

[✓ 13]

☐ 20

☐ The outcome depends on what `y` is bound to in the context

Consider the following program:

```
(let ((f (lambda (g)
           (lambda (y) (g 10)))))
  ((f (lambda (x) (+ x y))) 3))
```

In a language *with lexical scoping* (also known as *static scoping*) what would be the outcome of running the program?

Select one answer

☐ 3

☐ 13

☐ 20

[✓ The outcome depends on what `y` is bound to in the context]

Select the statement that most precisely summarizes how environments and substitutions traverse parts of the program during evaluation.

Select one answer

☐ Using environments we traverse parts of the program that may not actually be evaluated.

[✓ Using substitution we traverse parts of the program that may not actually be evaluated.]

☐ Using either environments or substitution there is no difference between how many times parts of the program will be traversed.

Say our goal is to define a datatype by cases, where one can add new cases to the datatype and new functions over the datatype, without recompiling existing code, and while retaining static type safety (e.g., no casts).

Select the statement that most accurately characterizes how *algebraic datatypes* vs. *objects* meet this goal.

Select one answer

☐ Using objects you can add new datatype cases, but not add new functions, without recompiling existing code.
Using algebraic datatypes you can add new functions, but not add new datatype cases, without recompiling existing code.

[✓ Using algebraic datatypes you can add new datatype cases, but not add new functions, without recompiling existing code.
Using objects you can add new functions, but not add new datatype cases, without recompiling existing code.]

☐ Using either algebraic datatypes or objects you can add new datatype cases, but not add new functions, without recompiling existing code.
Neither algebraic datatypes nor objects meet the goal of adding new functions, without recompiling existing code.

☐ Neither algebraic datatypes nor objects support adding new datatype cases nor functions, without recompiling existing code.