Question 1

Check a box if and only if it is an accurate description of ML

- ML uses lexical scope for the semantics of looking up variables in the environment
    - correct
- ML has no language constructs for creating mutable data
- ML has a REPL as part of the definition of the language
- ML is statically typed
    - correct

Question 2

Here is a particular list of pairs in ML:

CODE

```
[(4,19), (1,20), (74,75)]
```

For each pattern below, check the box if and only if this pattern matches the value above.

- `x::y`
    - correct
- `x::(y::z)`
    - correct
- `(a,b,c)::d`
- `[]`
- `(a,b)::(c,d)::(e,f)::[]`

- correct

- `(a,b)::(c,d)::(e,f)::g`
  - correct

Question 3

For each of the statements below, check the box if and only if the statement is true regarding this ML code:

CODE

```
fun mystery f xs =
    let
        fun g xs =
           case xs of
               [] => NONE
             | x::xs' => if f x then SOME x else g
    in
        case xs of
```

- `mystery` uses currying to take two arguments.
  - correct
- `mystery` uses tupling to take two arguments.
- If the second argument to `mystery` is a zero-element list, then whenever mystery produces a result, the result is `NONE` .
  - correct
- If the second argument to `mystery` is a one-element list, then whenever `mystery` produces a result, the result is `NONE` .
  - correct
- If the second argument to `mystery` is a two-element list, then whenever `mystery` produces a result, the result is `NONE` .
- The argument type of `f` can be any type, but it must be the same type as the element type of `xs` .

    - correct

- The result type of `f` can be any type, but it must be the same type as the element type of `xs` .

- If you replace the first line of the code with `fun mystery f = fn xs =>` , then some callers of `mystery` might no longer type-check.

- If you replace the first line of the code with `fun mystery f = fn xs =>` , then some callers of `mystery` might get a different result.

- `g` is a tail-recursive function.

    - correct

- For the entire computation of a call like `mystery someFun someList` , the total number of times `someFun` is called is always the same as the length of `someList` (for any `someFun` and `someList`).

- For the entire computation of a call like `mystery someFun someList` , the total number of times `someFun` is called is sometimes the same as the length of `someList` (for any `someFun` and `someList`).

    - correct

- For the entire computation of a call like `mystery someFun someList` , the total number of times `someFun` is called is never the same as the length of `someList` (for any `someFun` and `someList`).

Test code:

CODE

```
fun mystery f xs =
    let
        fun g xs =
           case xs of
               [] => NONE
             | x::xs' => if f x then SOME x else g
```

```
in
```

result:

CODE

```
val f = fn : int -> bool
val r1 = NONE : int option
val r2 = NONE : int option
val r3 = NONE : int option
val r4 = SOME 2 : int option
val r5 = SOME 2 : int option
```

Question 4

The `null` function is predefined in ML's standard library, but can be defined in many ways ourselves. For each suggested definition of `null` below, check the box if and only if the function would behave the same as the predefined null function whenever the function below is called.

Note: Consider only situations where calls to the functions below type-check.

- `fun null xs = case xs of [] => true | _ => false`
    - correct
- `fun null xs = xs=[]`
    - correct
- `fun null xs = if null xs then true else false`
- `fun null xs = ((fn z => false) (hd xs)) handle List.Empty => true`
    - correct

Question 5

The next four questions, including this one, relate to this situation: Suppose somebody has written a library for a collection of strings (perhaps implemented as some sort of linked list of strings or tree of strings, but the details do not matter). The library includes higher-order functions map, filter, and fold that operate on these collections and have their conventional meanings. For each problem below, decide which of these library functions is the best to use for implementing the desired function.

(For those needing a precise definition of best: On this exam, the best function, given appropriate arguments, returns the final result you need, meaning you need no more computation after calling the function. If multiple functions can do this, choose the one that can be used by passing it the function argument that itself does the least amount of work.)

Desired function: Take a collection of strings and produce a new collection where each string in the output is like a string in the input except the string has any space characters removed.

- map
    - correct
- filter
- fold

Question 6

Desired function: Take a collection of strings and return a string that is the concatenation of all the strings in the collection.

- map
- filter
- fold
    - correct

Question 7

Desired function: Take a collection of strings and a number n and return how many strings in the collection have a length that is a multiple of n.

- map
- filter
- fold
    - correct

Question 8

Desired function: Take a collection of strings and return a collection containing the strings in the input collection that start with a capital letter.

- map
- filter
    - correct
- fold

Question 9

This datatype binding and type synonym are useful for representing certain equations from algebra:

CODE

```
datatype algebra_exp = Variable of string
                     | Integer of int
                     | Decimal of real
                     | Addition of algebra_exp * a
                     | Multiplication of algebra_e
                     | Exponent of algebra_exp * i
type equation = algebra_exp * algebra_exp
```

Which of the mathematical equations below could not be elegantly represented by a value of type equation?

- $x + y = z$
- $(x + 4) + z = 7 \cdot y(x + 4) + z = 7 \cdot y$
- $x^3 . y^2 = z^0$
- $14.2 + 3 = 17.2$
- $x^y = z$
    - correct

Question 10

Here is a particular polymorphic type in ML:

CODE

```
'a * 'b -> 'b * 'a * 'a
```

For each type below, check the box if and only if the type is an instantiation of the type above, which means the type above is more general.

- `string * int -> string * int * int`
- `int * string -> string * int * int`
    - correct
- `int * int -> int * int * int`
    - correct
- `{foo : int, bar : string} -> {a : string, b : int, c : int}`
- `'a * 'a -> 'a * 'a * 'a`
    - correct

Question 11

The next 5 questions, including this one, are similar. Each question uses a slightly different definition of an ML signature `COUNTER` with this same structure definition:

CODE

```
structure NoNegativeCounter :> COUNTER =
struct

exception InvariantViolated

type t = int

fun newCounter i = if i < 0 then 1 else i
```

In each problem, the definition of `COUNTER` matches the structure definition `NoNegativeCounter`, but different signatures allow clients to use the structure in different ways. You will answer the same question for each `COUNTER` definition by choosing the best description of what it allows clients to do.In this question, the definition of `COUNTER` is:

CODE

```
signature COUNTER =
sig
    type t = int
    val newCounter : int -> t
    val increment : t -> t
    val first_larger : t * t -> bool
end
```

- This signature allows (some) clients to cause the `NoNegativeCounter.InvariantViolated` exception to be raised.
  - Correct, the method is to call `NoNegativeCounter.first_larger(~1, ~3)`
- This signature makes it impossible for any client to call `NoNegativeCounter.first_larger` at all (in a way that causes any part of the body of `NoNegativeCounter.first_larger` to be evaluated).

- This signature makes it possible for clients to call `NoNegativeCounter.first_larger` , but never in a way that leads to the `NoNegativeCounter.InvariantViolated` exception being raised.

Question 12

In this question, the definition of `COUNTER` is:

CODE

```
signature COUNTER =
sig
    type t = int
    val newCounter : int -> t
    val first_larger : t * t -> bool
end
```

- This signature allows (some) clients to cause the `NoNegativeCounter.InvariantViolated` exception to be raised.
    - Correct, the method is to call `NoNegativeCounter.first_larger(~1, ~3)`
- This signature makes it impossible for any client to call `NoNegativeCounter.first_larger` at all (in a way that causes any part of the body of `NoNegativeCounter.first_larger` to be evaluated).
- This signature makes it possible for clients to call `NoNegativeCounter.first_larger` , but never in a way that leads to the `NoNegativeCounter.InvariantViolated` exception being raised.

Question 13

In this question, the definition of `COUNTER` is:

CODE

```
signature COUNTER =
sig
    type t
    val newCounter : int -> int
    val increment : t -> t
    val first_larger : t * t -> bool
end
```

- This signature allows (some) clients to cause the `NoNegativeCounter.InvariantViolated` exception to be raised.
- This signature makes it impossible for any client to call `NoNegativeCounter.first_larger` at all (in a way that causes any part of the body of `NoNegativeCounter.first_larger` to be evaluated).
    - `t` Correct, because a variable of type cannot be instantiated
- This signature makes it possible for clients to call `NoNegativeCounter.first_larger`, but never in a way that leads to the `NoNegativeCounter.InvariantViolated` exception being raised.

Question 14

In this question, the definition of `COUNTER` is:

CODE

```
signature COUNTER =
sig
    type t
    val newCounter : int -> t
    val increment : t -> t
    val first_larger : t * t -> bool
end
```

- This signature allows (some) clients to cause the `NoNegativeCounter.InvariantViolated` exception to be raised.

- This signature makes it impossible for any client to call `NoNegativeCounter.first_larger` at all (in a way that causes any part of the body of `NoNegativeCounter.first_larger` to be evaluated).
- This signature makes it possible for clients to call `NoNegativeCounter.first_larger`, but never in a way that leads to the `NoNegativeCounter.InvariantViolated` exception being raised.
    - Correct, because `newCounter` negative numbers will not be generated and `first_larger` negative numbers cannot be called directly.

Question 15

In this question, the definition of `COUNTER` is:

CODE

```
signature COUNTER =
sig
    type t = int
    val newCounter : int -> t
    val increment : t -> t
end
```

- This signature allows (some) clients to cause the `NoNegativeCounter.InvariantViolated` exception to be raised.
- This signature makes it impossible for any client to call `NoNegativeCounter.first_larger` at all (in a way that causes any part of the body of `NoNegativeCounter.first_larger` to be evaluated).
    - correct because there is no `first_large`
- This signature makes it possible for clients to call `NoNegativeCounter.first_larger`, but never in a way that

leads to the `NoNegativeCounter.InvariantViolated` exception being raised.