Quiz 1

Questions 1 - 3 (6 possible versions each)

1.
```
fun foo (a, b) =
  if b = 0
   then a
   else foo (b, a mod b)
val x = x′
val y = ???
val ans = (foo (x, y) = z′)
```

x′	**z′**	**y**
35	7	any multiple of 7, but not a multiple of 5
35	5	any multiple of 5, but not a multiple of 7
35	1	any integer not a multiple of 5 or 7
12	6	any multiple of 6, but not a multiple of 12
12	3	any multiple of 3, but not a multiple of 4
12	1	any integer not a multiple of 2, 3, or 4

2.
```
val x = 2
val y = ???
val q =
    let
        val x = 7
        val z = x + z′
    in
        x + y - z
    end
val ans = (q = q′)
```

z′	**q′**	**y**
1	2	3
1	5	6
1	8	9
2	0	2
2	2	4
2	7	9

3.
```
fun baz (x, lst) =
    let
     fun help (n, l) =
         case l of
              [] => NONE
            | head::tail => if head = x
                            then SOME n
                            else help (n + 1, tail)
    in
     help (0, lst)
    end
val x = ???
val y = y′
val ans = (baz(x, y) = z′)
```

y′	**z′**	**x**
`[4, 8, 15, 16, 23, 42]`	`SOME 0`	4
`[4, 8, 15, 16, 23, 42]`	`SOME 3`	16
`[4, 8, 15, 16, 23, 42]`	`NONE`	1 (or any number not in the list)
`[8, 6, 7, 5, 3, 0, 9]`	`SOME 3`	5
`[8, 6, 7, 5, 3, 0, 9]`	`SOME 5`	0
`[8, 6, 7, 5, 3, 0, 9]`	`NONE`	1 (or any number not in the list)

Questions 4 - 6

4.
```
(* evaluates to SOME v where v is the first negative number
 * in lst, or NONE there are no negative numbers in lst *)
fun first_negative lst =
    case lst of
        [] => NONE
      | head::tail => if head < 0
                      then head
                      else first_negative tail
```

   a) *Types of branches don't match; evaluating to `int option` in empty case but `int` in non-empty case*
   b)
```
fun first_negative lst =
    case lst of
        [] => NONE
      | head::tail => if head < 0
                      then SOME head
                      else first_negative tail
```

5.
```
(* sums the first element of each list in xs *)
fun sum_heads xs =
    case xs of
        [] => 0
      | x::xs' => x + sum_heads xs'
val ans = sum_heads [[1, 2], [3, 4, 5], [6]]
```

   a) *Trying to add `int list` to an `int` in the non-empty case*
   b)
```
fun sum_heads xs =
    case xs of
        [] => 0

      | []::xs' => sum_heads xs'
      | (x::_)::xs' => x + sum_heads xs'
val ans = sum_heads [[1, 2], [3, 4, 5], [6]]

fun sum_heads xs =
    case xs of
        [] => 0

      | []::xs' => sum_heads xs'
      | x::xs' => hd x + sum_heads xs'
val ans = sum_heads [[1, 2], [3, 4, 5], [6]]
```

6.
```
datatype food =
        Pizza of string
      | Burger of int * bool
      | Salad

(* determines whether a food is healthy (Salad) or not (Pizza and
 * Burger) *)
fun is_healthy f =
   case f of
      Pizza => false
     | Burger => false
     | Salad => true
```

a) *Constructors* `Pizza` *and* `Burger` *in patterns are missing arguments*

b)
```
fun is_healthy f =
   case f of
      Pizza _ => false
     | Burger _ => false
     | Salad => true
```

Questions 7 - 8 (2 possible versions each)

```
7. fun bar lst =
      case lst of
         [] => 0
        | NONE::tail => bar tail
        | SOME n::tail => n + (bar tail)
```

a) *Computes the sum of all the `SOME` elements in the argument*

b)
```
fun sum_somes_tail lst =
     let
       fun loop (lst, acc) =
            case lst of
              [] => acc
              | NONE::tail => loop(tail, acc)
              | SOME n::tail => loop(tail, n + acc)
     in
       loop(lst, 0)
     end
```

```
fun bar lst =
   case lst of
      [] => 0
     | NONE::tail => 1 + (bar tail)
     | _::tail => bar tail
```

a) *Counts the number of `NONE` elements in the argument*

b)
```
fun count_nones_tail lst =
     let
       fun loop (lst, acc) =
            case lst of
              [] => acc
              | NONE::tail => loop(tail, 1 + acc)
              | _::tail => loop(tail, acc)
     in
       loop (lst, 0)
     end
```

```
8. fun foo (strs, sep) =
       case strs of
         [] => ""
        | s::[] => s
        | s::strs' => s ^ sep ^ foo(strs', sep)
```

a) *Concatenates the elements of `strs` with `sep` between each*

b)
```
fun concat_with_tail (strs, sep) =
     let
      fun loop (strs, acc) =
          case strs of
            [] => acc
            | [s] => acc ^ s
            | s::ss' => loop (ss', acc ^ s ^ sep)
     in
      loop (strs, "")
     end
```

```
fun foo nums =
   case nums of
     [] => 0
    | [n] => n
    | x::y::tail => x + (foo tail)
```

a) *Sums every other element in the argument*

b)
```
fun sum_every_other_tail nums =
     let
      fun loop (nums, acc) =
          case nums of
            [] => acc
            | [n] => n + acc
            | x::y::tail => loop(tail, x + acc)
     in
      loop (nums, 0)
     end
```

Questions 9 - 10

For the next two questions, recall the following code from lecture:

```
(* a datatype to represent arithmetic expressions *)
datatype exp =
      Const of int
      | Negate of exp
      | Add of exp * exp
      | Mult of exp * exp

(* evaluates its argument to produce an integer result *)
fun eval e =
    case e of
     Const i => i
      | Negate e1 => ~ (eval e1)
      | Add (e1, e2) => (eval e1) + (eval e2)
      | Mult (e1, e2) => (eval e1) * (eval e2)
```

Question 6 (4 possible versions)

9. Write an expression to go in place of `???` below so that `ans` will be bound to ***z′*** after the given code is executed. Assume the datatype `exp` and the function `eval` are bound.

```
val x = ???
val y = Add(x, Negate(Mult(Const a′, Const b′)))
val ans = eval y
```

z′	**a′**	**b′**	**x**
15	3	~2	Const 9
15	~1	3	Const 12
23	4	~3	Const 11
23	~1	3	Const 20

10. Write a function `remove_add_zeroes` that has type `exp -> exp` that returns its argument, but with all instances of adding an expression to `Const 0` removed.

```
fun remove_add_zeroes e =
    case e of
     Add (Const 0, e2) => remove_add_zeroes e2
      | Add (e1, Const 0) => remove_add_zeroes e1
      | Add (e1, e2) => Add (remove_add_zeroes e1,
                             remove_add_zeroes e2)
      | Mult (e1, e2) => Mult (remove_add_zeroes e1,
                               remove_add_zeroes e2)
      | Negate e1 => Negate (remove_add_zeroes e1)
      | _ => e
```