Quiz 2 - OCaml

Q1 OCaml Typing
4 Points

• For the following two sub-questions, you are not allowed to use type annotations.
• All pattern matching must be exhaustive.
• No other warnings should be raised.

Q1.1 Ocaml Typing
2 Points

Write an OCaml expression of type `int -> string -> bool`

fun x y -> x = int_of_string y

Q1.2 OCaml Typing
2 Points

Write an OCaml expression of type `('a -> 'b) -> 'a -> 'c -> 'b`

fun x y z = z y

Q2 OCaml Typing2
4 Points

Q2.1 OCaml Typing2
2 Points

Write the type of the following expression

```
let rec f p x y =
match x, y with
|[],[] -> []
|(a,b)::t1, c::t2 -> (p a c, p b c)::(f p t1 t2)
|_, _ -> failwith "error"
```

('a -> 'b -> 'c) -> ('a * 'a) list -> 'b list -> ('c * 'c) list

Q2.2 OCaml Typing2
2 Points

Write the type of the following expression

```
let f p x y = map (p x) y
```

('a -> 'b -> 'c) -> 'a -> 'b list -> 'c list

Q3 Fill in the Blank
6 Points

Given the following `fold` implementation, implement a function called `min_and_max which given a list of integers between 1 and 100 inclusive returns a tuple whose first value is the minimum value in the list and whose second value is the maximum value in the list. You can assume the list will not be empty.

```
let rec fold f a xs = match xs with
|[] -> a
|x::xt -> fold f (f a x) xt
```

Examples:

```
min_and_max [4;5;3;8] = (3, 8)
min_and_max [10; 23; 5; 79] = (5, 79)
```

Note: You are not allowed to use the `List` module.

Prompt:

```
let min_and_max lst = fold (_____#1_____) (_____#2_____) lst
```

Blank #1:

(fun (a,b) c ->
let min = if a < c then a else c in
let max = if b > c then b else c in
(min,max))

Blank #2:

(100,1)

Q4 OCaml Coding
6 Points

Write a function "sumtiply," which returns a new list of multiplications between sum of all elements of lst1 and each element of lst2.

- You can assume that lst1 will not be empty.
- You can create helper functions but the `rec` keyword should not be used.
- You can use the following definitions of Map and Fold, but you cannot use the `List` module

```
let rec map f l = match l with
[]-> []
|h::t -> (f h)::(map f t);;

let rec fold f a l = match l with
[]->a
|h::t -> fold f (f a h) t;;
```

For Example:

```
sumtiply [1;2;3;4;5] [1;3;5;7;9] = [15;45;75;105;135]
```

let sumtiply l1 l2 = let sum = fold (+) 0 l1 in map (fun x -> sum * x) l2