CMSC330 Fall 2024 Quiz

Proctoring TA: ____________ **Name:** ____________

Section Number: ____________ **UID:** ____________

Problem 1: Basics

[Total 4 pts]

	True	False
In OCaml, all values are expressions but not all expressions are values	T	F
Having mutable variables can make it hard to reason about how a program runs	T	F
`let f x = x + 3` is an example of a higher order function	T	F
An OCaml function can return different types depending on how it's called	T	F

Problem 2: OCaml Typing and Evaluating

[Total 6 pts]

Give the type for the following functions `f` and give what the following function call evaluates to. **If there is a type error in the function**, put "TYPE ERROR" for the type, and put "ERROR" for the evaluation. If the function call does not follow the type of `f`, put "ERROR" for the evaluation.

(a) [2 pts]

```
let f x y = match x with
   [] -> []
  |x::xs -> y :: xs ;;

f [] [1;2;3] ;;
```

Type of f:

Evaluation:

(b) [2 pts]

```
let f a b =
    if b > 5 then a
    else true ;;

f 2.0 false;;
```

Type of f:

Evaluation:

(c) [2 pts]

```
let rec f g lst = match lst with
   [] -> []
  |x::xs -> (x, g x)::(f g xs) ;;

f (fun x -> x mod 2 = 1) [1;2;3] ;;
```

Type of f:

Evaluation:

Problem 3: OCaml Expressions [Total 4 pts]

Write an expression that would have the following types.

(a) [2 pts]

```
'a -> 'a -> bool list
```

(b) [2 pts]

```
('a -> 'a) -> 'a -> int
```

Problem 4: Coding [Total 6 pts]

Write a function encode that takes a `int list` and returns a `string list`, which consists of the string "1" repeated by each number in the int list. You may assume that all values in the input list are >= 0.

You **do NOT have to use map or `fold`**, but their definitions are given if you want to use them.
You can write helper methods.

```
(* Examples
    encode [0;1;2;3] = ["";"1";"11";"111"]
    encode [0;0;3] = ["";"";"111"]
*)


(* Write your code below *)
```

```
let rec map f l = match l with
   [] -> []
  |x::xs -> (f x)::(map f xs)

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

```
 let rec encode lst =
```