1. Give the type of the following OCaml expressions. If any has an error, describe it.
   a. `[("1", 2); ("3", 4)]`
   b. `fun f a -> [a; a + 1]`
   c. `fun x y -> [x ; y]`

2. Give the value of the following OCaml expressions. If any has an error, describe it.
   a. `[1; 2]::[3]`
   b. `let x y = y 3 in x (fun z -> z - 1)`
   c. `map ((fun x y -> x+y) 1) [2; 3; 4]`

3. Write an OCaml expression with the type of each of the following:
   a. `int * int list`
   b. `int list -> (int -> int)`
   c. `(int -> 'a) -> 'a`

4. Solve the following OCaml programming problems. You are allowed to use `List.rev` (an OCaml library function that reverses a list) and the following curried versions of the function map and fold, but no other OCaml library functions. Your solution must run in $\mathcal{O}(n)$ time for input lists of length n.

```
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;;
```

   a. Write a function `makeLists` that when applied to a list `lst`, creates a new list for every element of `lst`, returning the results in a single list. You may use map or fold if you wish, but it is not required. Example: `makeLists [1; 2; 4]` should produce `[[1]; [2]; [4]]`.

   b. Using either map or fold and an anonymous function, write a function `over20` which when applied to a list of ints `lst`, returns a list of all elements of lst that are 21 or over (preserving their relative order in `lst`). Example: `over20 [33; 18; 21; 19]` should produce `[33; 21]`.

   c. Write a function `growNum` which when given a number `k`, returns a list of the first `k` integers, starting from `k`. Examples:
   `growNum 0` should return `[]`
   `growNum 1` should return `[1]`
   `growNum 4` should return `[1; 2; 3; 4]`

   d. Using either map or fold and an anonymous function, write a curried function `countNum` which when given a number `n` and a list of ints `lst`, returns the number of times `n` occurs in `lst`.
   Examples:
   `countNum 5 [1; 2; 3; 4]` should return `0`
   `countNum 5 [1; 2; 3; 4; 5; 6]` should return `1`
   `countNum 5 [5; 5; 6; 5]` should return `3`