1. (4 pts) Give the types of the following OCaml expression
   a. (2 pts) \([1.0];[2.0;3.0]\) \hspace{1cm} \text{Type = float list list}
   b. (2 pts) \text{let } f \ (\_::\_\_\_) = x;; \hspace{1cm} \text{Type = 'a list -> 'a}

2. (3 pts) Write an expression of type \text{int -> int -> int}

   \text{let } f \ x \ y = x+y;;
3. (4 pts) Write a recursive function `sumSmall` which takes in an int list `lst` and an integer threshold `x` and recursively sums up the elements of `lst` which are strictly less than `x`. For instance, given the list [1;2;1;4;2;3] 3, `sumSmall` will return 6.

```ocaml
let rec sumSmall lst x = match lst with
  (h::t) -> if (h<x) then h + (sumSmall t x) else (sumSmall t x)
| [] -> 0
;;
```

4. (4 pts) Using map or fold and an anonymous function, write an Ocaml function `timesThree`, which takes in a list of floats `lst` and returns a list of floats in which each element is 3 times greater. For instance, calling `timesThree` on [1.0; 2.0; 3.0] would return [3.0; 6.0; 9.0]. If you do not use map or fold, you will not receive credit.

```ocaml
let timesThree lst = map (fun x -> x *. 3.0) lst;;
```