COMP 302 W25 Practice Problem Set 1

Problem 1: Type Inference (Higher-Order Functions)

a) Infer the type of:

```
let rec twist lst acc =
  match lst with
  | [] -> acc
  | h :: t -> twist t (h :: acc)
```

b) Infer the type of:

```
let mystery f g x = match x with
  | Some y -> f y
  | None -> g ()
```

Problem 2: Tracing Expressions with Shadowing

a) Trace:

```
let x = 5 in
let f y = x + y in
let x = 10 in
f x
```

b) Trace `sum [1; 2; 3]`:

```
let sum lst =
  let rec helper acc lst =
    match lst with
    | [] -> acc
    | h :: t -> helper (acc + h) t
  in helper 0 lst
```

c) Trace:

```
let x = 2 in
let f g = g (x + 1) in
let x = 5 in
f (fun y -> x * y)
```

Problem 3: Environment Lookup and Evaluation

a) Implement `lookup : string -> (string * float) list -> float option` that searches an association list for the first occurrence of a variable. Below are a few examples:

- `lookup "x" [("x", 2); ("y", 3)]` ⇒ `Some 2`
- `lookup "z" [("x", 2)]` ⇒ `None`
- `lookup "x" [("x", 1); ("x", 2)]` ⇒ `Some 1`

b) Given the types of `expr` and `env` shown below. Implement the function `eval : expr -> env -> float option` where the evaluation *fails* if an unknown variable is encountered. (Hint: Use `lookup` you developed in part a)).

```
type expr =
  | Const of float
  | Var of string
  | Plus of expr * expr
  | Times of expr * expr

type env = (string * float) list
```

Problem 4: Structural transformation of Part 3

a) Rewrite `lookup` to use CPS with two continuations. Observe that since lookup was already TR, the continuations in the CPS version of lookup don't change between recursive calls.

b) Rewrite `eval` to use `mystery` from problem 1b instead of explicit pattern-matching.