6. Debugging, Opsems, Types (18 points)

Consider these shown Reason programs that *do not type-check*; the code implicated by the type checker will be highlighted and underlined. Each has English comments explaining what the program *should* do, as well as assertions that *should* type check *and* succeed.

(a)
```
/* "sepConcat sep [s1;s2;s3]" should insert "sep" between "s1", "s2", and "s3", and
    concatentate the result. */
/* Recall that List.fold_left takes a function, an accumulator, and a list as input */
let rec sepConcat = fun sep sl =>
  switch sl {
  | [] => ""
  | [h, ...t] =>
    let f = fun a x => a ^ (sep ^ x);
    let base = [];
    List.fold_left f base sl
  };

assert (sepConcat "," ["foo", "bar", "baz"] == "foo,bar,baz");
```

i. (3 points) Why is `sepConcat` not well-typed?

ii. (3 points) Describe how you would fix the code so that `sepConcat` works correctly.

(b)
```
/* "padZero xs ys" returns a pair "(xs', ys')" where the shorter of "xs" and "ys" has
    been left-padded by zeros until both lists have equal length. */
let rec clone = fun x n =>
  if (n <= 0) {
    []
  } else {
    [x, ...clone x (n - 1)]
  };

let padZero = fun l1 l2 => {
  let n = List.length l1 - List.length l2;
  if (n < 0) {
    (clone 0 ((-1) * n) @ l1, l2)
  } else {
    (l1, [clone 0 n, ...l2])
  }
};

assert (padZero [1, 2] [1] == ([1, 2], [0, 1]));
```

i. (3 points) Why is `padZero` not well-typed?

ii. (3 points) Describe how you would fix the code so that `padZero` works correctly.

(c)
```
/* "mulByDigit d [n1;n2;n3]" should multiply the "big integer" "[n1;n2;n3]"
   by the single digit "d". */
let rec mulByDigit = fun d n =>
  switch (List.rev n) {
  | [] => []
  | [h, ...t] => [mulByDigit d t, (h * d) mod 10]
  };

assert (mulByDigit 4 [2, 5] == [1, 0, 0]);
```

i. (3 points) Why is `mulByDigit` not well-typed?

ii. (3 points) Describe how you would fix the code so that `mulByDigit` works correctly.