Lab 10: Local and lambda

Create a separate file for each question. Keep them in your "Labs" folder, with the name **`l`**`ij`**`q`**`k` for **Lab** `ij`, **Question** `k`.

Download the headers for each function from the file `l10-interface.rkt`.

After you have completed a question (except class exercises), including creating tests for it, you can obtain feedback by submitting it and requesting a public test. Follow the instructions given in the Style Guide.

This lab makes use of the following data definitions:

```
;; A Tweet is a (list Str Str (listof Str)), where
;; * The first value is the name of the tweeter, starting with @
;; * The second value is message, containing at most 280 characters
;; * The third value is a list of hashtags, each starting with #
```

Language level: Intermediate Student with lambda

Important note: Do not write **any** explicitly recursive code in your solutions. Each solution should use at least one of the following abstract list functions: `map`, `filter`, `foldr`, `build-list`, `sort`, `andmap`, `ormap`. You may also find `range` useful.

1. *[Class exercise with lab instructor assistance]*

A prefix of a string s is a substring of s starting from position 0. Create a function *prefixes* that consumes a string, *str*, and produces a list of all non-empty prefixes of *str*.

For example:

```
(prefixes "abcd") => (list "a" "ab" "abc" "abcd")
```

2. Complete a function *powers-of-two* that consumes a natural number, *n*, and produces a list of powers of 2, from 2^n down to $2^0 = 1$.

   For example:

   ```
   (powers-of-two 2) => (list 4 2 1)
   ```

3. *[Part a]* Complete a function *mult-table* that consumes two natural numbers, *width* and *height*, and produces a multiplication table of size *width* by *height* as a list of lists. The first element of the answer will be a list that is the first row of the multiplication table, and so on. This multiplication table will include a row/column for zero.

   For example:

   ```
   (mult-table 3 4) => (list (list 0 0 0)
                             (list 0 1 2)
                             (list 0 2 4)
                             (list 0 3 6))
   ```

[Part b] Complete the function *times-table*, that works like *mult-table*, but this time there will **not** be a row/column for the products of zero. You are not allowed to use the built-in functions `remove` or `remove-all` for this.

For example:

```
(times-table 3 4) => (list  (list 1 2 3)
                            (list 2 4 6)
                            (list 3 6 9)
                            (list 4 8 12))
```

4. Create a function *count-suffixes* that consumes a list of strings, *los*, and a string, *suffix*, and produces the number of strings in *los* which end with *suffix*. This is similar to `count-starters` from lecture.

   For example:

   ```
   (count-suffixes (list "howdy" "there" "how" "are") "re")
   => 2
   ```

5. Create a function *matching-messages* that consumes a list of Tweets, *tweets*, and a string that starts with #, *hashtag*, and produces the list of strings corresponding to the messages in tweets whose hashtag list contains *hashtag*. Note the data definition of a Tweet included above.

   For example:

   ```
   (define some-tweets
     (list (list "@julia" "I am very excited"
                 (list "#bikeride" "#nohomework"))
           (list "@jack" "Frustrating..."
                 (list "#notsunny" "#toomuchhomework"))
           (list "@jessica" "What should I do?"
                 (list "#nohomework" "#raining"))))
   (matching-messages some-tweets "#nohomework")
   => (list "I am very excited" "What should I do?")
   ```

Optional open-ended questions

Return to previous problems from labs, assignments, and lecture and identify where `local` and `lambda` might have been appropriate.

Helpful tips

Testing local helpers

Some local helpers might be complicated enough (`insert`, for example), that we want to test them on their own. However, you cannot use `check-expects` inside of a `local`. A possible strategy is to temporarily move this helper outside of the local, to the top level. There, you can test the function using `check-expect`. When testing is complete, move the function back into the `local` and delete the tests and examples.