Q 1

Define a recursive function `REM-FIRST`, which removes the first occurrence of its first argument from the second.

Q 2

Define a recursive function `D-HOW-MANY?` that counts all – not only top-level – the occurrences of an item in a list.

Q 3

The built-in function `LAST`, when applied to a list, returns the final element of the list in a list – try and see. Define a recursive function `RLAST`, which returns the final element – the element itself not a list containing it, if there is one; return `nil`, otherwise.

Q 4

Define a recursive function `CHOP-END`, which removes the final element of the given list – its like `CDR` from the back. You are NOT allowed to make `(REVERSE (CDR (REVERSE LST)))`. Nothing to be done for an empty list, just return it as it is; but a single element list gets "nilled".

Q 5

Define a recursive function `SUM` that sums the integers in its list argument.

Q 6

Define a recursive function `PALINDROME` that checks whether a given list is a palindrome. You are NOT allowed to use `REVERSE` this time, you need to think recursively. You may need to use some functions you defined above.

Q 7

Define a recursive function `C-LENGTH`, custom version of the built-in `LENGTH`. You will need to keep a counter that increases in every recursion. The counter should start as `0`; therefore make your function 2-place (=getting two arguments as inputs), such that it is always called with `0` as the second argument. What will be your base case? What you will return when you arrive there? We did this in class; solve it without looking at the notes.

Q 8

Define a recursive function `P-SUM`, which takes a list of two element lists of integers, e.g. `((7 8) (1 13) (4 1))`, and returns their sum, 34 for this case.

Q 9

Define a recursive function `L-PROD`, which takes an integer and a list of integers, returns a list of integers where each element in the second list got multiplied by the first integer argument.

Q 10

Define a recursive function `PAIR-PROD`, which takes a list of two element lists of integers as above, but this time returns a list of products of these pairs. E.g. an input like `((7 8) (1 13) (4 1))` should yield `(56 13 4)`.

Q 11

Define a recursive function `BLANK-N` , which takes an integer `N` and a list, and replaces every `N`th element of the list with the symbol `X`.

```
* (blank-n 3 '(1 3 4 5 1 5 4))

(1 3 X 5 1 X 4)
```

Q 12

Define a recursive function `MULTI-MEMBER` that checks if its first argument occurs more than once in the second.

Q 13

Define a recursive function `REM-LAST`, which removes the last occurrence of its first argument from the second.

Q 14

Define a recursive function `SUBSTITUTE`, with 3 arguments, say `old new exp` such that every occurrence of `old` at the top-level of `exp` is replaced by `new`. By "top-level" we mean the function should not check embedded levels in lists. E.g. `(substitute 'x 'k '(x (x y) z))` should return `(k (x y) z)`.

Q 15

Modify `SUBSTITUTE` to `D-SUBS` (for "deep substitute"), so that it does the replacement for *all* occurrences of `old`, no matter how deeply embedded.