

Repetition with Recursion

- Exercises
 - Exercise 0: Preparation
 - Exercise 1: Product
 - Exercise 2: Squaring Lists
 - Exercise 3: Lengths of Lists
 - Exercise 4: How Much Did You Skip?
 - Exercise 5: No Skipping Allowed
 - Exercise 6: Tallying Symbols
 - Exercise 7: Counting Odd Numbers
 - Exercise 8: Extracting Odds
 - Exercise 9: Closest to Zero
 - Exercise 10: Finding the Gaps
 - Exercise 11: Are They All Valid?
 - Exercise 12: Is It There?
 - Exercise 13: Types
 - Exercise 14: How Many Values?

Exercises

Exercise 0: Preparation

- a. Reflect on the key aspects of recursion.
- b. Start DrScheme.

Exercise 1: Product

Define and test a Scheme procedure, (`product values`), that takes a list of numbers as its argument and returns the result of multiplying them all together. Warning: (`product null`) should *not* be 0. It should be the identity for multiplication, just as (`sum null`) is the identity for addition. Explain why.

Exercise 2: Squaring Lists

Define and test a Scheme procedure, (`square-each-element values`), that takes a list of numbers as its argument and returns a list of their squares.

```
> (square-each-element (list -7 3 12 0 4/5))  
(49 9 144 0 16/25)
```

Hint: For the base case, consider what the procedure should return when given a null list; for the other case, separate the car and the cdr of the given list and consider how to operate on them so as to construct the desired result.

Exercise 3: Lengths of Lists

Define and test a Scheme procedure, (`lengths lists`) that takes a list of lists as its argument and returns a list of their lengths:

```
> (lengths (list (list 'alpha 'beta 'gamma)
                 (list 'delta)
                 null
                 (list 'epsilon 'zeta 'eta 'theta 'iota 'kappa)))
(3 1 0 6)
```

Exercise 4: How Much Did You Skip?

Define and test a Scheme procedure, (`tally-skips lst`), that takes one argument, a list, and determines how many times the symbol `skip` occurs in the list.

Exercise 5: No Skipping Allowed

Define and test a Scheme procedure, (`filter-out-skips lst`), that takes a list of symbols as its argument and returns a list that does not contain the symbol `skip`, but is otherwise identical to the given list. (Use the predicate `eq?` to test whether two symbols are alike.)

```
> (filter-out-skips (list 'hop 'skip 'jump 'skip 'and 'skip 'again))
(hop jump and again)
```

The example illustrates the intended effect of the procedure. By itself, however, it's not an adequate test of your procedure. It would be a good idea to test the case in which the given list is empty, a case in which it contains only `skips`, and one in which it contains only symbols other than `skip`. You might also test different positions of `skip`: at the front, at the end, and in the middle.

We recommend that you test the procedures you create very thoroughly. In most cases, testing does not reveal any errors in your procedures; but finding and correcting the errors that testing exposes is one of the most productive and rewarding uses of a programmer's time.

Exercise 6: Tallying Symbols

Define and test a Scheme procedure, (`tally-occurrences sym symbols`), that takes two arguments, a symbol and a list of symbols, and determines how many times the given symbol occurs in the given list.

Hint: Use direct recursion. Here are the questions that you must resolve: What is the base case? What value should the procedure return in that case? How can you simplify the problem in order to recursively invoke the procedure being defined? What do you need to do with the value of the recursive procedure call in order to obtain the final result?

```
> (tally-occurrences 'apple (list 'pear 'apple 'cranberry 'banana 'apple))
2
> (tally-occurrences 'apple (list 'oak 'elm 'maple 'spruce 'pine))
0
```

Exercise 7: Counting Odd Numbers

Write a Scheme procedure, (`num-odds values`), that returns the number of odd numbers in a list.

If you'd like to be extra careful, make sure that your procedure works correctly even if some of the values in the list are not numbers.

Exercise 8: Extracting Odds

Write a Scheme procedure, (`odds values`), that, given a list, produces another list that contains only the odd numbers in the first list.

Exercise 9: Closest to Zero

Write a Scheme procedure, (`closest-to-zero values`), that, given a list of numbers (including both negative and positive numbers), returns the value closest to zero in the list.

Exercise 10: Finding the Gaps

Define and test a Scheme procedure, (`gaps values`), that takes a non-empty list of real numbers as its argument and returns a list of the disparities between numbers that are adjacent on the given list.

```
> (gaps (list 30 16 21 9 42))
(14 5 12 33)
```

Note that `gaps` always returns a list one element shorter than the one it is given.

Hint: What is the base case?

Exercise 11: Are They All Valid?

Define and test a Scheme predicate, (`all-in-range? values`), that takes a list as argument and determines whether all of its elements are in the range from 0 to 100, inclusive.

Exercise 12: Is It There?

Define and test a Scheme predicate, (`element? sym symbols`), that takes two arguments, a symbol and a list, and determines whether the given symbol is an element of the given list.

Exercise 13: Types

Define and test a Scheme procedure, `(types lst)`, that takes a list as an argument and returns a list of the types of the individual elements.

For example,

```
> (types '(34 a (1 2 3)))
(number symbol list)
> (types '())
()
> (types '(a b c))
(symbol symbol symbol)
```

You may want to use your `type` procedure from a previous lab.

Exercise 14: How Many Values?

This problem is for extra credit.

Some of you have criticized the build-in `length` method for counting only the number of values in the top-level list. Write a procedure, `count-values` that counts the total number of non-list values in a list or its sublists.

For example,

```
> (count-values '(a b c))
3
> (count-values '())
0
> (count-values '((1 2 3 (4 5)) (a b) ((2))))
8
> (count-values '(() () () ()))
0
```