

Symbols and Lists

In this lab, you will explore the basic operations for working with lists and symbolic values. You may want to refer to the corresponding reading as you work on this lab.

Procedures used in this lab: `append`, `car`, `cdr`, `cons`, `length`, `list`, `list-ref`, and `reverse`.

- Exercises
 - Exercise 0: Preparation
 - Exercise 1: Two simple lists
 - Exercise 2: Interpreting compound conses
 - Exercise 3: Using the `list` procedure
 - Exercise 4: Building a list of symbols
 - Exercise 5: Empty lists
 - Exercise 6: Repeated elements
 - Exercise 7: A small `cdr`
 - Exercise 8: Extracing information from empty lists
 - Exercise 9: Extracting information from symbols
 - Exercise 10: Compound lists
 - Exercise 11: It's all Greek to me
 - Exercise 12: How long is emptiness?
 - Exercise 13: Checking your own length
 - Exercise 14: Length of compound lists
 - Exercise 15: Reversing lists
 - Exercise 16: Reversing compound lists
 - Exercise 17: Joining lists
 - Exercise 18: Listing lists
 - Exercise 19: Consing lists
 - Exercise 20: Extracing elements
 - Exercise 21: Wrapup
- Notes
 - Notes on Problems 18 and 19

Exercises

Exercise 0: Preparation

Start DrScheme.

Exercise 1: Two simple lists

- Call the `cons` procedure to create a list that has the number 1 as its first and only element. The result of your call should be `(1)`.
- Call the `cons` procedure to create a list that has the symbols `a` and `b` as its two elements. The result of your call should be `(a b)`.

Exercise 2: Interpreting compound `conses`

- Figure out (without using DrScheme) the result of the following expression.

```
(cons 'alpha (cons 'beta (cons 'gamma (cons 'delta null))))
```

- Check your answer by asking DrScheme to evaluate this expression.

Exercise 3: Using the `list` procedure

Call the procedure `list`, supplying the numerals 17 and 43 as arguments. Describe the value returned by the procedure.

Exercise 4: Building a list of symbols

- How would you call the `list` procedure to create a list containing the symbols `alpha`, `beta`, and `gamma`, in that order?
- Verify your answer by entering the code in DrScheme.

Exercise 5: Empty lists

How would you invoke the `list` procedure to create an empty list?

Exercise 6: Repeated elements

Determine by experiment whether it is possible to create a list in which the same element occurs more than once.

Exercise 7: A small `cdr`

- What is the `cdr` of a one-element list?
- Verify your answer by experimentation.

Exercise 8: Extracing information from empty lists

It makes no sense to apply the `car` and `cdr` procedures to an empty list, because there's no way to split off the "first element" of a list that has no elements. What happens if you try it anyway? Find out by having DrScheme evaluate a deliberately incorrect procedure call.

Exercise 9: Extracting information from symbols

- Does it make sense to apply `car` and `cdr` to values other than lists? Why or why not?
- Determine what happens if you apply these procedures to symbolic values and numeric values.

Exercise 10: Compound lists

- Create the list `(e)`
- Create the list `(d (e))`
- Create the list `(b c)`
- Create the list `(a (b c) (d (e)))`

Exercise 11: It's all Greek to me

Use Scheme to give the name `Greek-letters` to the list constructed by the expression `(list 'alpha 'beta (list 'gamma-1 'gamma-2) 'delta)`. Then call the `length` procedure to confirm that it has four elements.

Exercise 12: How long is emptiness?

Determine the length of the empty list.

Exercise 13: Checking your own length

- Create a list of length 5. I don't care what's in the list.
- Check your answer by having Scheme compute the length of that list.

Exercise 14: Length of compound lists

- Create the list `(a (b c) (d (e)))`
- What do you think the length of this list should be?
- Experimentally determine the length of this list.

d. Explain the result.

Exercise 15: Reversing lists

Use Scheme to compute the reversal of the list whose elements are the symbols `senior`, `junior`, `sophomore`, and `freshling`, in that order.

Exercise 16: Reversing compound lists

- If a list has another list as one of its elements, should `reverse` reverse that inner list as well as the outer one?
- Find out by experiment what Scheme does.

Exercise 17: Joining lists

Use Scheme to find the result of stringing together (with `append`) a list with the symbols `alpha` and `beta` as its elements and a list with the numbers 1, 2, and 3 as its elements. How many elements does the resulting list have?

Exercise 18: Listing lists

- Invoke the procedure `list`, applying it to the two lists that you strung together in the previous exercise: a list with the symbols `alpha` and `beta` as its elements and a list with the numbers 1, 2, and 3 as its elements.
- How many elements does the resulting list have?
- The answer to this question is different from the answer to the question at the end of the previous exercise -- why?

Exercise 19: Consing lists

- Write a call to the procedure `cons`, applying it to our favorite two lists: a list with the symbols `alpha` and `beta` as its elements and a list with the numbers 1, 2, and 3 as its elements.
- How many elements does the resulting list have?
- Why is the answer to this question different from the answers to the questions at the end of the previous two exercises.

Exercise 20: Extracing elements

Write a call to the `list-ref` procedure that will extract the fourth element of the list

(38 72 apple -1/3 sample)

That is, you should extract the number $-1/3$.

Exercise 21: Wrapup

Quit DrScheme and log out of the workstation.

Notes

Notes on Problems 18 and 19

The `append` procedure joins together the elements of a list to make a new list. Hence, when you append two lists together, the total number of elements in the new list is the sum of the number of elements in the lists.

The `list` procedure creates a new list whose elements are the parameters to `list`. Hence, if `list` takes two parameters, the length of the result is always two, regardless of what those parameters are.

The `cons` procedure builds a new list by placing its first parameter at the start of its second parameter (which is a list). Hence, the length of the result is one more than the length of the second parameter.