Carleton University
Department of Systems and Computer Engineering
SYSC 3101 - Programming Languages - Winter 2018

Lab 2 - Processing Lists in Scheme/Racket

References

Structure and Interpretation of Computer Programs

- Section 2.1.1 - the subsection titled *Pairs* describes `cons`, `car` and `cdr`
- Section 2.2 up to the end of Section 2.2.1, describes how lists are represented and processed in Scheme/Racket

Two documents at the Racket website provide plenty of information about the Racket dialect of Scheme:

The Racket Guide, https://docs.racket-lang.org/guide/index.html

- See Section 3.8, *Pairs and Lists*

The Racket Reference, https://docs.racket-lang.org/reference/index.html

- Section 4.9, *Pairs and Lists*, summarizes the Racket procedures that operate on immutable lists and pairs.

A guide to the DrRacket IDE can be found here:

http://docs.racket-lang.org/drracket/index.html

Racket Coding Conventions

Please adhere to the conventions described in the Lab 1 handout.

Getting Started

Launch the DrRacket IDE.

If necessary, configure DrRacket so that the programming language is Racket. To do this, select Language > Choose Language from the menu bar, then select The Racket Language in the Choose Language dialog box.

`#lang racket` should appear at the top of the definitions area. Don't delete this line.

"The Rules"

Do not use special forms that have not been presented in lectures. Specifically,

- Do not use `set!` to perform assignment; i.e., rebind a name to a new value.
- Do not use any of the Racket procedures that support *mutable* pairs and lists (`mpair`, `mcons`, `mcar`, `mcdr`, `set-mcar!`, `set-mcdr!`), as described in Section 4.10 of *The Racket Reference*.
- Do not use `begin` expressions to group expressions that are to be evaluated in sequence.

You can use `lambda` expressions to create procedures and `let` expressions to create local variables, but they aren't required.

You are allowed to use the procedures that are described in these sections of *The Racket Reference*:

- Section 4.9.1, *Pair Constructors and Selector*
- Section 4.9.2, *List Operations*
- Section 4.9.6, *Pair Accessor Shorthands*
- Section 4.9.7, *Additional List Functions and Synonyms*: `empty`, `cons?`, `empty?`, `first`, `rest`, `second` through `tenth`, `last`, `last-pair`.

Unless otherwise noted, you are **not** allowed to use the procedures that are described in these sections of *The Racket Reference*:

- Section 4.9.3, *List Iteration*
- Section 4.9.4, *List Filtering*
- Section 4.9.5, *List Searching*
- Section 4.9.7, *Additional List Functions and Synonyms*: with the exception of the permitted procedures listed earlier.
- Section 4.9.8, *Immutable Cyclic Data*

You can save your solutions to the exercises in a single file; for example, `lab2.rkt`, or you can create a different file for each exercise.

Exercise 1

Part (a) Define procedure `(sum-numbers numbers)`. It takes a list of numbers as an argument and returns their sum. Do not use Racket's `apply` procedure to apply `+` to the list. Your procedure must recursively sum the numbers.

Part (b) Define procedure `(average numbers)`. It takes a list of numbers and returns their average. This procedure must call the `sum-numbers` procedure you defined for part (a).

Exercise 2

Define procedure `(occurrences numbers n)`. It takes a list of numbers and a number, and calculates how many times the number occurs in the list. For example:

```
> (occurrences '(1 3 5 2 7 5 8 9 5) 5)

3
```

Remember, you are not allowed to use any of Racket's list searching procedures (*The Racket Reference*, Sections 4.9.5 and 4.9.7).

Hint: review the `contains?` procedure posted on cuLearn before you attempt this exercise.

Exercise 3

Define procedure `convert`. It takes a list of decimal digits and produces the corresponding integer number. The first digit in the list is the least significant. For example:

```
> (convert (cons 1 (cons 2 (cons 3 empty))))

321

> (convert (list 4 5 6))

654

> (convert '(2))

2
```

Exercise 4

Define procedure `convertFC`. It takes a list of temperature measurements in degrees Fahrenheit and returns a list of the equivalent Celsius temperatures. Feel free to define "helper" procedure(s) that are called by `convertFC`.

Remember, you are not allowed to use `map` or any of the other list iteration procedures provided by Racket (*The Racket Reference*, Sections 4.9.3 and 4.9.7).

Exercise 5

Define procedure `eliminate-threshold`. This procedure takes a list of numbers and a threshold value. It returns a list containing all numbers that are below or equal to the threshold. For example,

```
> (eliminate-threshold (list 3 7 0 4 1 5) 4)

'(3 0 4 1)

> (eliminate-threshold (list 3 7 0 4 1 5) -1)

'()  ; returns the empty list
```

Remember, you are not allowed to use `map`, `filter`, or any of the other "forbidden" procedures listed in *"The Rules"* on Page 2.

Revised Feb. 8, 2018: In Exercises 1 and 2, changed the name of parameter `list` to `numbers`. (`list` will be bound to a list of numbers when the procedure is called, which means the function cannot call Racket's `list` procedure).