Carleton University
Department of Systems and Computer Engineering
SYSC 3101 - Programming Languages - Winter 2019

Lab 1 - Introduction to Racket (Scheme)

References

Evans, *Introduction to Computing*, Chapter 3. (The URL for this book is in the course outline.)

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

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

A guide to the DrRacket IDE can be found here:

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

Racket Coding Conventions

Indentation: Racket code is formatted differently than Python or C programs. Please adhere to the indentation style used in Evans' book and the lectures. DrRacket provides a command to reformat code in the definitions area (Racket > Reindent All).

Naming constants: Don't litter your code with literal values; instead, use `define` to give names to frequently-used constants, and use the names instead of the literal values in your procedures. For example, in a procedure that converts inches to cm, you should have this definition:

```
(define CM-PER-INCH 2.54)
```

and use `CM-PER-INCH` in the procedure.

On the other hand, there's little point in this definition:

```
(define ONE 1)

(define (add-one x)
  (+ x ONE))
```

Replacing `1` with `ONE` doesn't make the `add-one` procedure easier to understand.

Procedure names: A common convention for choosing a procedure name is to use a noun or noun-phrase that describes what the procedure returns. This makes expressions that apply the procedure easier to read. For example, use `discounted-price` instead of `calculate-discounted-price`, or `days-remaining` instead of `determine-days-remaining`.

There are exceptions to this convention. For example, one of the lecture examples was this procedure:

```
(define (improve guess x)
  (average guess (/ x guess)))
```

The procedure's name is a verb, `improve`, instead of a noun phrase; for example, `improved-guess`. The call `(improve guess x)` implies that the procedure returns an improved guess. Would `(improved-guess guess x)` make the code easier to understand?

Predicate names: Predicates are procedures that return boolean values (true or false). The names of these procedures should end in `?`; for example, `odd?` or `good-enough?`.

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 `let` expressions to create local variables.
- Do not use `begin` expressions to group expressions that are to be evaluated in sequence.

When defining procedures, you can use either `lambda` expressions or Scheme's condensed notation for procedure definitions. For example, you can define a square procedure this way:

```
(define square (lambda (x) (* x x)))
```

or this way (condensed notation):

```
(define (square x)
  (* x x))
```

Exercise 1

Type these procedures in the *definitions window*:

```
;; Calculate the area of a circle with the specified radius.
(define (area-of-circle radius)
  (* pi radius radius))

;; Calculate the area of a ring whose radius is outer and
;; whose hole has a radius of inner.
(define (area-of-ring outer inner)
  (- (area-of-circle outer)
     (area-of-circle inner)))
```

Click the Save button to save the definitions in a file (for example, lab1.rkt).

Click Run to evaluate the code in the definition window and make the definitions available in the *interactions window*.

To interactively test a procedure, we type an expression that *applies* the procedure to its argument(s).

Test `area-of-circle` by typing this expression after the `>` prompt:

```
> (area-of-circle 5)
```

The result is displayed below the expression. For this test, the value should be approximately `78.5`.

Test `area-of-ring` by typing this expression:

```
> ( area-of-ring 10 5)
```

The result should be approximately `235.6`.

You can save your solutions to Exercises 2 through 7 in the same file you created in Exercise 1, or you can create a different file for each exercise (select File > New Tab from the menu bar to open a new definitions pane).

Exercise 2

When we click DrRacket's Run button, it determines if the definitions are well-formed. If a definition is ill-formed, DrRacket displays a message that describes the syntax error.

The three procedures in this exercise are intended to add 10 to their argument; however, each definition has a syntax error.

Type this procedure in the definitions area, exactly as shown here:

```
(define (f 1)
  (+ x 10))
```

Click Run and read the error message. Fix the definition.

Type this procedure in the definitions area, exactly as shown here:

```
(define (g x)
  + x 10)
```

Click Run and read the error message. Fix the definition.

Type this procedure in the definitions area, exactly as shown here:

```
(define h(x)
  (+ x 10))
```

Click Run and read the error message. Fix the definition.

Exercise 3

The local supermarket needs a program that can compute the value of a bag of coins. Define procedure `sum-coins`. It consumes four numbers: the number of pennies, nickels, dimes, and quarters in the bag. Its result is the amount of money in the bag, as a quantity of pennies.

Test your procedure by typing these expressions in the interactions window:

```
(sum-coins 1 0 0 0)    ; result should be 1
(sum-coins 0 1 0 0)    ; result should be 5
(sum-coins 0 0 1 0)    ; result should be 10
(sum-coins 0 0 0 1)    ; result should be 25
(sum-coins 1 1 1 1)    ; result should be 41
```

Exercise 4

In this exercise, you're going to define two versions of a procedure that calculates the surface area of a solid cylinder. It consumes the radius and height of the cylinder.

1. Define a procedure named `area-of-cylinder-one-def` that doesn't call any helper procedures; in other words, all the operators are primitive procedures provided by Racket (e.g., `+`, `-`, `*`, `/`).

   Test your procedure by typing these expressions:

   ```
   (area-of-cylinder-one-def 2 3) ; result should be about 62.8
   (area-of-cylinder-one-def 3 4) ; result should be about 131.9
   ```

2. Define a procedure named `area-of-cylinder`. Define helper procedures with descriptive names that are called by `area-of-cylinder` (in the same way that the `area-of-ring` procedure in Exercise 1 calls `area-of-circle`).

   Test your procedure by typing these expressions:

   ```
   (area-of-cylinder 2 3) ; result should be about 62.8
   (area-of-cylinder 3 4) ; result should be about 131.9
   ```

Which version of the procedure is easier to understand?

Exercise 5

Define procedure `interest`. It consumes a bank account balance (the amount of money in the account), and calculates the amount of interest that the money earns in one year. The bank pays a flat 4% for balances up to $1,000, a flat 4.5% per year for balances above $1000 and up to $5,000, and a flat 5% for balances of more than $5,000. Hint: use a `cond`-expression. See the lecture slides for an example.

Test your procedure by typing these expressions:

```
(interest 500)   ; result should be 20
(interest 1000)  ; result should be 40
(interest 2000)  ; result should be 90
(interest 5000)  ; result should be 225
(interest 10000) ; result should be 500
```

Exercise 6

A quadratic equation has the form $ax^2 + bx + c = 0$. The equation's coefficients are a, b and c, and variable x represents the unknown. A value of x is a solution to the equation if, for specific coefficients, $ax^2 + bx + c$ evaluates to 0.

The number of solutions a quadratic equation has depends on the values of the coefficients. If coefficient a is 0, we say the equation is *degenerate* and do not consider how many solutions it has.

Assuming a is not 0, the equation has

- two solutions if $b^2 > 4 \cdot a \cdot c$,
- one solution if $b^2 = 4 \cdot a \cdot c$, and
- no solution if $b^2 < 4 \cdot a \cdot c$.

(The expression $b^2 - 4 \cdot a \cdot c$ is called the *discriminant*.)

To distinguish this case from the degenerate one, we sometimes use the phrase *proper* quadratic equation.

Define procedure `how-many-solutions`, which consumes the coefficients `a`, `b`, and `c` of a proper quadratic equation and determines how many solutions the equation has.

Test your procedure by typing these expressions:

```
(how-many-solutions 1 2 1)  ; result should be 1
(how-many-solutions 2 4 1)  ; result should be 2
(how-many-solutions 2 4 3)  ; result should be 0
(how-many-solutions 1 0 -1) ; result should be 2
(how-many-solutions 2 4 2)  ; result should be 1
```