Carleton University
Department of Systems and Computer Engineering
SYSC 3101 - Programming Languages - Winter 2020
Lab 1 - Introduction to Racket (Scheme)

References

David Evans, *Introduction to Computing*, Chapter 3. http://computingbook.org
Two documents at the Racket website provide plenty of information about the Racket dialect of Scheme:
The Racket Guide, https://docs.racket-lang.orqg/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: 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.

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))
```

Add:

```
(require 2htdp/image)
```

to the beginning of the file. These commands tell Racket that you want to use two pre-existing software packages, one specifically for this class, and another that's a general graphics library. Now hit the **Run** button. That will cause Racket to load those packages in and let you use them.

To call procedures, Racket uses a uniform notation for procedure calls:

```
(procedure operand1, operand2, ... )
```

Different procedures require different numbers and kinds of inputs. For example, the rectangle procedure requires two numbers (the width and height) followed by two strings (the "drawing mode" – either "solid" or "outline", and the name of the color to use). So we can make a 50x50 blue rectangle by saying:

```
(rectangle 50 50 "solid" "blue")
```

Try running this now. Now try changing the arguments to different values: change the width and height, the color, etc.

All the procedures in Racket are explained in the Racket documentation. You can access the documentation from Help>**Racket Documentation** or by right clicking on any word in your program and selecting **"Search in Help Desk for …".**

Look up the documentation on the **rectangle**. See how it explains the inputs (it calls them arguments). The funny notation, like "(and/c real? (not/c negative?))", is just a way of saying "a possibly fractional number that isn't

negative". You can click on "mode?" to see more about what drawing modes are allowed. Now look up the documentation for the **ellipse** procedure.

Part 1

Create a picture that looks like the following:

A 100x100 red square

Now you should name the square. In the definitions window type:

```
(define a-red-square (your-code-for-the-square)
```

and hit run.

You can now refer to the square by typing a-red-square into the REPL

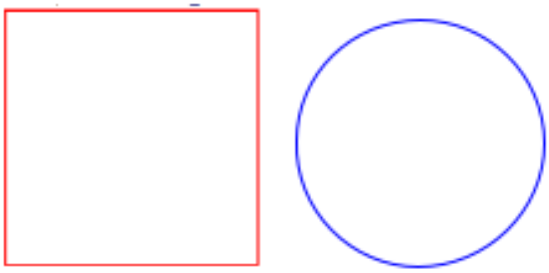
Part 2

A blue circle of radius 50:

Define it as the name: a-blue-circle

Part 3

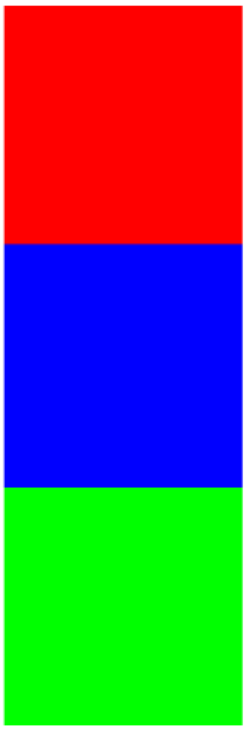
A red square and a blue circle in an outline form

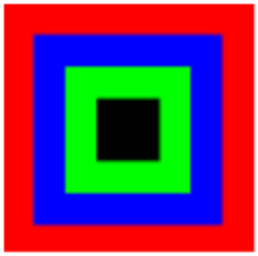


Define them as the names outlined-square and outlined-circle, respectively.

Part 4

Now read the documentation for **overlay**, **above** and **beside,** then create and combine 3 images to form the images below:

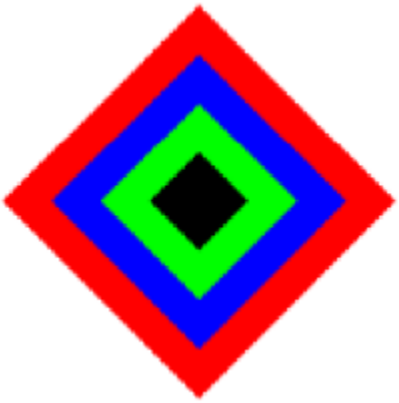




Define these as **row-of-squares, column-of-squares**, and **nested-squares** respectively.

Part 5

Read the documentation about rotate and try to make an image like this:



Define it as **rotated-square**

Part 6

Try your best to create canada flag!