Lab 1 – Introduction to Scheme

Some of the following exercises are taken from Chapter 1 of the book *Concrete Abstractions*. You may wish to refer to the chapter if needed while working through the exercises.

1. Define a procedure called `cube` that takes a number as input and computes its cube.

2. Define a procedure called `sphere-volume` that takes the radius of a sphere as input and computes its volume using the formula *Volume* = $(4/3)\ \pi\ r^3$ where π is the value 3.14159. Some examples are shown below:

```
(sphere-volume 1)   =>  4.188786666666666
(sphere-volume 3)   =>  113.09723999999999
(sphere-volume 0)   =>  0
```

3. Write a procedure `(distance x1 y1 x2 y2)` that takes the coordinate values of two points (*x1*, *y1*) and (*x2*, *y2*) as input, in that order, and computes the distance between the points using the formula below:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Remember that Scheme has a built-in procedure for computing square roots called `sqrt`. Test your procedure on the following input values:

```
(distance 1 2 4 6)    =>  5
(distance 1 1 2 2)    =>  1.4142135623730951
(distance 4 0 -6 3)   =>  10.44030650891055
```

4. Write a procedure called `average` that takes three numbers and returns their average.

5. Write a procedure called `middle3-average` that takes *five* numbers and returns the average of the middle three (dropping the highest and lowest values). Hint: use `min` and `max`.

6. Remember the good old quadratic formula from high school math, which can be used to figure out the value of *x* in the equation $ax^2 + bx + c = 0$? Here it is:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Actually, there are always two values of *x* (called "roots") that will satisfy the above equation. The symbol $\pm$ in the formula means that you can calculate the first root by using addition (+) in the formula and the second root by using subtraction (**-**). Write a procedure called `quadratic+` that takes the values *a*, *b*, and *c* as input and calculates the first root of the equation (the one using addition). Here are some examples:

```
(quadratic+ 2 3 1)   => -1/2
(quadratic+ 3 6 3)   => -1
(quadratic+ 5 6 2)   => -3/5+1/5i
```

7. Assume that you know how much money you typically spend daily (on food and other daily needs), weekly (on transportation costs, entertainment, etc.), and monthly (rent, loans, credit cards, etc.). Write a procedure called `annual-spending` to compute your typical annual spending. Your procedure should have three input parameters corresponding to the daily, weekly, and monthly spending amounts. There are 52 weeks in a year, and a total of 365 days (ignoring leap years).

8. Consider the "mystery" procedure below. Can you figure out what it does? Make some predictions about how it will behave on various input values, and then <u>carefully</u> type it in and try it out to verify your answers.

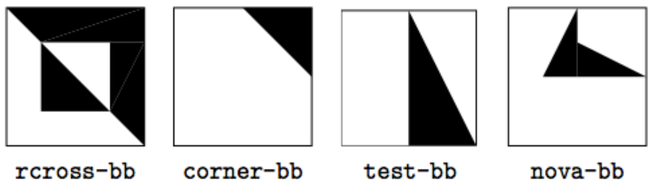
```
(define mystery
  (lambda (a b)
    ((if (> b 0) + -) a b)))
```

9. According to the *Joy of Cooking*, candy syrups should be cooked 1 degree cooler than listed in the recipe for each 500 feet of elevation above sea level. Define a procedure called `candy-temp` that takes two values as input: the recipe's temperature in degrees and the elevation in feet. It should calculate the temperature to use at that elevation. The recipe for Chocolate Caramels calls for a temperature of 244 degrees; suppose you wanted to make them in Denver, the "mile high city". (One mile equals 5280 feet.) Use your procedure to find the temperature for making the syrup.

10. Candy thermometers are usually calibrated only in integer degrees, so it would be handy if your `candy-temp` procedure would give an answer rounded to the nearest degree. Rounding can be done using the predefined procedure called `round`. For example, `(round 7/3)` and `(round 5/3)` both evaluate to 2. Insert an application of `round` at the appropriate place in your procedure definition and test it again.

11. The U.S. tax code uses what is called a *marginal tax rate*. This policy means roughly that the tax rate used depends on the level of income. For example, suppose that the first $10,000 of a person's income is not taxed at all, but the amount above $10,000 is taxed at 20 percent. If you earned $12,500, the first $10,000 would be untaxed, but the amount over $10,000, namely $2,500, would be taxed at 20 percent, yielding a tax bill of 20/100 × $2,500 = $500. Write a procedure called `tax` that takes a person's income and calculates the tax using these assumptions. Hint: use an `if` expression.

12. The next few exercises use the predefined graphics images shown below:



IMPORTANT: to use these images, you need to have installed the "concabs" library by following the instructions on our class web page. You must also include the following line at the top of your code:

```
(require (lib "fungraph.ss" "concabs"))
```

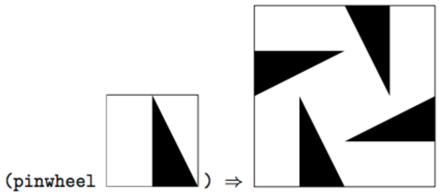
Try evaluating the following expressions, which use the operations `stack` and `quarter-turn-right` to combine images into more complex images in various ways:

```
(stack rcross-bb corner-bb)

(quarter-turn-right test-bb)

(stack (stack rcross-bb corner-bb) test-bb)

(stack (stack rcross-bb corner-bb)
       (stack (quarter-turn-right test-bb) test-bb))
```

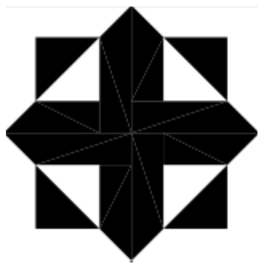
13. Define procedures `half-turn` and `quarter-turn-left` that do as their names suggest. Both procedures take a single argument, namely, the image to turn. You will naturally need to use the built-in procedure `quarter-turn-right`.

14. Define a procedure `side-by-side` that takes two images as arguments and creates a composite image having the first image on the left and the second image on the right. Hint: use the `quarter-turn-left` procedure you defined above to help you out.

15. Each dark cross in the repeating crosses pattern is formed by joining together four copies of the basic block, each facing a different way. We can call this operation *pinwheeling* the basic block; here is an example of the same operation performed on the image `test-bb`:



Define the `pinwheel` procedure and use it to make a cross as shown below:



16. Now try pinwheeling the cross—you should get a sample of the quilt, with four dark crosses, as shown on page 15 of *Concrete Abstractions*. If you pinwheel that, how big is the quilt you get? Try making other pinwheeled quilts in the same way, but using the other basic blocks. What do the designs look like?

17. All four of the basic blocks can be produced using two primitive graphics procedures. The first of these, `filled-triangle`, takes six arguments, which are the `x` and `y` coordinates of the corners of the triangle that is to be filled in. The coordinate system runs from -1 to 1 in both dimensions. For example, here is the definition of `test-bb`:

```
(define test-bb (filled-triangle 0 1 0 -1 1 -1))
```

The second of these procedures, `overlay`, combines images. To understand how it works, imagine having two images on sheets of transparent plastic laid one on top of the other so that you see the two images together. For example, here is the definition of `nova-bb`, which is made out of two triangles:

```
(define nova-bb
   (overlay (filled-triangle 0 1 0 0 -1/2 0)
            (filled-triangle 0 0 0 1/2 1 0)))
```

Use these primitive graphics procedures to define the other two basic blocks, `rcross-bb` and `corner-bb`.

18. Now that you know how it is done, be inventive. Come up with some basic blocks of your own and make pinwheeled quilts out of them. You might find it interesting to try experiments such as overlaying rotated versions of an image on one another.