Problem Set 1
Posted: Sept. 11, 2008
Due Back: Sept. 30, 2008

Problem 1.1. (20 pts)
a. For a given value of n, give an explanation of how many distinct regular n-gons can be produced. (The n-gons may have lines that cross each other, but each line must be of the same length and each corner angle must be equal.) Using your explanation, describe how many distinct 59-gons can be drawn by an expression of the following form:

(repeat 59 (fd 20) (rt x))

How many distinct 60-gons are there? (Note also that two n-gons differing only by a rotation are not considered “distinct” for the purposes of this question.)

b. Prove that all of the corner points on a regular n-gon drawn by an expression of the form

(repeat n (fd side) (rt theta))

lie on a circle.

Problem 1.2 (15 pts)
a. In the polygon procedure shown in lecture, the procedure arguments specify the number of sides of the polygon, and the length of an individual side. Thus, evaluating

(polygon 4 100)

produces a square each of whose sides is of length 100. Suppose that instead we would like a procedure that creates a regular polygon of a given number of sides and a given perimeter. A call to our new procedure (we'll name it polygon-perimeter) would appear as follows:

(polygon-perimeter 4 100)

Evaluating this expression would produce a square each of whose four sides is of length 25; similarly,

(polygon-perimeter 5 100)

would cause the turtle to draw a pentagon each of whose 5 sides is of length 20.

Write the polygon-perimeter procedure and test it out to convince yourself that it works.

b. Using the polygon-perimeter procedure, and noting that (i) we can make a good approximation to a circle by a 360-sided polygon, and that (ii) the perimeter of a circle is 2π times the radius of the circle, write a procedure draw-circle-with-radius. This procedure should take a single argument corresponding to the radius of the desired circle; when the procedure is called on that argument, the turtle should draw a circle of the given radius. Thus, evaluating

(draw-circle-with-radius 10)

will cause the turtle (starting from its current position and heading) to draw a circle with a radius of 10 units.

c. Suppose we wish to create a procedure that will draw a polygon of n sides, circumscribed about a circle of a given radius. (Note that the circle inscribed in the given polygon will touch the polygon at the center of
Write a procedure named `draw-circumscribing-polygon` which, when called on arguments `n-sides` and `radius`, will draw a polygon of `n` sides that circumscribes a circle of the given `radius`. Important: this procedure should assume that the turtle begins at the center of the inscribed circle, and should be state-invariant. Here, for instance, is what you would see after evaluating:

```
(draw-circumscribing-polygon 4 40)
(draw-circumscribing-polygon 6 40)
(draw-circumscribing-polygon 10 40)
```

Problem 1.3 (25 pts)
In lecture the Koch snowflake curve was generated by a recursive procedure: each nth-level snowflake was the result of pasting together four 1/3-scaled copies of the (n-1)-level snowflake:

```
(snowflake 60 0)
(snowflake 60 1)
```

Try creating a couple of variations of this idea. For instance, suppose we write a procedure whose 0th-level shape is just a straight line segment, and whose 1st-level shape is created by five 1/3-scaled copies of the segment as follows:
Write the `boxflake` procedure and try it at levels of 4 and 5. How does the number of sides of the shape vary with the level argument? Would calling the procedure with a level-value of (say) 20 be feasible? What is the similarity dimension of the (limiting, "infinite-level") boxflake curve.

Create at least one other variation of the snowflake idea using some other strategy of "pasting together" scaled-down versions of the previous-level shape at each new level. If your variation is self-similar, give its similarity dimension; if it is not, explain why the construction does not produce a self-similar shape.

**Problem 1.4 (25 pts)**

In class, we saw a recursive shape that produced a design like the following:

Here are the three (Scheme) procedures used to create this design:
(define (shape1 side level)
  (cond ((= level 0) 0)
        (else
         (fd (/ side 2))
         (rt 45)
         (shape2
          (* (sqrt 2) (/ side 2)) (- level 1))
         (lt 45)
         (bk (/ side 2))
         (repeat 4
          (shape3 (/ side 4) (- level 1))
          (fd side) (rt 90)))))

(define (shape2 side level)
  (cond ((= level 0) 0)
        (else
         (repeat 2
          (shape3 (/ side 2) (- level 1))
          (fd (/ side 2))
          (shape1 (/ side 2) (- level 1))
          (fd (/ side 2))
          (rt 90)
          (fd side)
          (rt 90)))))

(define (shape3 side level)
  (cond ((= level 0) 0)
        (else
         (repeat 4
          (fd (/ side 3))
          (shape1 (/ side 3) (- level 1))
          (fd (* 2 (/ side 3)))
          (rt 90)))))

Your job is to make up your own set of mutually-recursive procedures to create a pleasing design.
Problem 1.5. (15 pts)
Implement the `cornerpoly` procedure described in Abelson and diSessa (pp. 89-90), and vary it in some interesting way. For example, I implemented the `cornerpoly` procedure, but also included some variation in the choice of "turtle-move" so that the turtle could move forward as a basic step, but could also move in (say) a zig-zag pattern. Examples are shown below. You needn't imitate this variation, but you should play with `cornerpoly` to produce something interesting.

```
(define (zig side)
  (fd side) (rt 144)
  (fd (/ side 2)) (lt 144) (fd side))

(cornerpoly 40 60 zig 0)
```
(define (flag side)
  (fd side) (repeat 3 (fd side) (rt 120)) (fd (* 2 side)))

(cornerpoly 40 60 flag 0)