Stopping Recursive Procedures

It can be annoying when your recursive procedure keeps going and going. You might want it to stop at a certain point when you've got your design the way you want it. For example, let's look at our `polyspi` function:

```
to polyspi :side :angle
fd :side
rt :angle
polyspi :side + 10 :angle
end
```

You might not want it to fill the whole screen (or blot out the whole screen if you are in `wrap` mode). So you can do this:

```
to polyspi2 :side :angle :times
      if less? :times 1 [stop]
      fd :side
      rt :angle
      polyspi2 :side + 10 :angle :times - 1
end
```

In `polyspi2`, we added a new variable called `:times`. Notice the line

```
      polyspi2 :side + 10 :angle :times - 1
```

Every time we invoke `polyspi2` from inside `polyspi2`, we add 10 to `:side` and subtract 1 from `:times`.

As we keep subtracting from `:times`, `:times` will eventually be smaller than 1. That's where the following line kicks in:

```
      if less? :times 1 [stop]
```

The `less?` operation needs two inputs. In this case, the inputs are `:times` and `1`. The `less?` operation compares its two inputs, and if `:times` is less than 1, it outputs `"true`. Otherwise, it outputs `"false`.

The `if` command also takes two inputs. The first input it needs is either `"true` or `"false`. The second input is a list of instructions. This can be any list of instructions, like the list of instructions you use as input to `repeat`.

In our case, the instruction is `stop`. This tells Logo to exit the procedure without going any further.

So, when `:times` is less than one, the procedure stops and recursion ends. How many times will `polyspi2` call itself if we start with `polyspi2 30 40 20`?

Try using this trick with some of your other recursive functions.

Drawing Polygons with Repeat

We know that in order to draw a polygon with `:n` sides, a turtle has to turn `360 / :n` degrees on each turn. So we can write

```
to polygon :n
repeat :n [fd 100 rt 360 / :n]
end
```

We can also add a variable for the length of each side:

```
to ngon :n :edge
repeat :n [fd :edge rt 360 / :n]
end
```

We can use ngon to write a cool procedure from Jim Clayson's book, *Visual Modeling with Logo*:

```
to spingon :n :edge :angle :growth
ngon :n :edge
rt :angle
spingon :n (:edge * :growth) :angle :growth
end
```

Try these:

```
? window
? spingon 30 2 10 1.02 95
? cs repeat 3 [spingon 4 120 0 0.95 50 rt 90]
? spingon 4 120 0.95 19
```

We can also add a `:times` variable and an `if` statement so we can stop `spingon` after a certain number of times:

```
to spingon :n :edge :angle :growth :times
if :times < 1 [stop]
ngon :n :edge
rt :angle
spingon :n (:edge * :growth) :angle :growth :times - 1
end
```

Recursive Designs

You’ve drawn designs using recursive procedures, but there’s a whole other class of designs that Harold Abelson and Andrea diSessa call recursive designs “because these figures contain subparts which are in some sense equivalent to the entire figure”.

Here are some recursive designs from their book, *Turtle Geometry*.

The Nested Triangle

This procedures starts by drawing a triangle. Then, each of corner of the triangle, it draws a smaller triangle. Then, in the corner of each of those triangles, it draws a smaller triangle, and so on.

```
to nested.triangle :size
if :size < 10 [stop]
repeat 3 [nested.triangle :size / 2 fd :size rt 120]
end
```

What makes this recursive procedure stop?

Snowflake

These procedures draw “a triangle in which each side is made up of four subsides, each subside is made up of four sub-subsides, etc.”

```
to snowflake :size :level
repeat 3 [side :size :level rt 120]
end

to side :size :level
if :level = 0 [fd :size stop]
side :size / 3 :level - 1
lt 60
side :size / 3 :level - 1
rt 120
side :size / 3 :level - 1
lt 60
side :size / 3 :level - 1
end
```

Try these

```
? cs snowflake 200 2
? cs snowflake 200 4
```

Cool Curves

Here are two more cool designs from *Turtle Geometry* by Abelson and diSessa.

The first one is the C curve:

```
to c :size :level
if :level = 0 [fd :size stop]
c :size :level - 1
rt 90
c :size :level - 1
lt 90
end
```

Try

```
? cs c 5 10
```

According to Abelson and diSessa, “a level 0 C curve is just a line; a level n C curve consists of two level n – 1 C curves at right angles to each other, followed by a 90 degree turn to restore the heading.”

Here are the procedures for drawing a dragon curve:

```
to ldragon :size :level
if :level = 0 [fd :size stop]
ldragon :size :level - 1
lt 90
rdragon :size :level - 1
end

to rdragon :size :level
if :level = 0 [fd :size stop]
ldragon :size :level - 1
rt 90
rdragon :size :level - 1
end
```

Try

```
? cs ldragon 5 11
? cs rdragon 5 11
```

Now What?

You've done an amazing job this week. Here are some suggestions for those of you who would like to do more with Logo.

You can get your own home version of Berkeley Logo for free from Brian Harvey's home page:

http://www.cs.berkeley.edu/~bh/

If you have a PC at home, I recommend that you use George Mills' MSWLogo, which is also free. There's a link to the MSWLogo site on Brian Harvey's home page.

The web site above also has a free online version of Brian Harvey's classic three volumes, *Computer Science Logo Style*. These texts cover computer science and Logo in depth.

Approaching Precalculus Mathematics Discretely by Philip G. Lewis is an excellent book that covers a lot of the math used in computer graphics. In addition to working with trig functions, you'll get to do linear algebra. Both trig and linear algebra are important for work in 2D and 3D graphics. This book is out of print, but internet booksellers like Amazon.com usually have a copy.

What about other computer languages? I recommend that students work with Logo at least through middle school and well into high school. After you have a really good grasp of Logo (i.e., after working through a substantial amount of Brian Harvey's books on Logo) a next good language would be Scheme, which is closely related to Logo and is commonly used to teach computer science to college students (Scheme is used in introductory and other computer science courses for computer science majors at Northwestern).

Program according to your own interests. We've done lots of graphics programming, but you can also use Logo to do natural language processing or write games. You can find a variety of interesting projects to work on in Brian Harvey's books.

I had a lot of fun working with you guys. Enjoy your further explorations of Logo or whatever else you decide to explore!

—TJ