Procedures within procedures

Once you have created your own procedure, like `square` or `triangle`, you can use the procedure name just like you use the names of primitives like `rt`, `show`, `fd`, or `heading`. You can even use them inside other procedures. In the example below, you can see a `square` procedure, a `triangle` procedure, and another procedure that uses those procedures.

```
to square
repeat 4 [fd 50 rt 90]
end

to triangle
repeat 3 [fd 50 rt 120]
end]

to thingy
square
fd 80
lt 40
triangle
end
```

What do you think `thingy` will do? Define these procedures and see what happens when you execute `thingy` (You *execute* an instruction when you type it at the `?` prompt and press press enter).

Make your own procedures that use square, triangle, or other procedures you have created. You might try representational drawings (drawings that are pictures of something, like a picture of a house) or abstract drawings (designs), or both.

Spin a path

A *path* is any line you draw with the turtle. It might be a *closed path*, like a square or a triangle, but it doesn't have to be (When a turtle walks a closed path, it gets back to its original heading and position).

To spin the path, you can do something like this:

```
to spin.thingy
repeat 50 [thingy rt 30]
end
```

To execute a `spin.thingy` instruction, you would type `spin.thingy` at the `?` prompt and press return. From now on, I'll write something like this:

? `spin.thingy`

when I mean "type `spin.thingy` at the `?` prompt and press return."

This is kind of boring, because you can only make one design. If you want to experiment, you can add a variable for the number of times you turn `thingy`:

```
to spin.thingy2 :times
repeat :times [thingy rt 30]
end
```

To execute a `spin.thingy2` instruction, you could type something like

? `spin.thingy2 10`

to turn `thingy` ten times.

This is a little better, but I would also add a variable for the angle you turn each time:

```
to spin.thingy3 :times :angle
repeat :times [thingy rt :angle]
end
```

For this one, you could try

```
? spin.thingy3 3 120
? spin.thingy3 5 72
```

I encourage you to make up your own designs to spin and to try out numbers of your own for `:times` and `:angle`. It's really fun to come up with your own designs, and you learn interesting things about paths and angles.

Make Wallpaper

Take a path and plop it all over the screen in different places. There are different ways to do this. Suppose you already have a `square` procedure (if you don't, you need to write one). You could write a procedure like this:

```
to wallpaper
repeat 25 [pd square pu rt 15 fd 60 lt 15]
end
```

To execute a `wallpaper` instruction, you would type `wallpaper` at the `?` prompt and press return. From now on, I'll write something like this:

```
?wallpaper
```

when I mean "type `wallpaper` at the `?` prompt and press return."

What do you do if that's not quite the effect you want? Add some variables and play around with them. For example, you could have a variable for the number of times you want to draw the square:

```
to wallpaper2 :times
repeat :times [pd square pu rt 15 fd 60 lt 15]
end
```

To execute a `wallpaper2` instruction, you could type something like

```
?wallpaper2 100
```

You could also have a variable for the distance to go forward:

```
to wallpaper3 :times :distance
repeat :times [pd square pu rt 15 fd :distance lt 15]
end
```

For this one, you could try

```
? wallpaper3 100 120
```

And you could add a variable for the angle of turn:

```
to wallpaper4 :times :distance :angle
repeat :times [pd square pu rt :angle fd :distance lt :angle]
end
```

Notice that we added `:angle` twice in the body of the procedure—once after `rt` and once after `lt`. Why? Of course, you don't have to wallpaper your screen with squares. Instead of the `square` procedure, you can use any procedure that draws a path.

Play with Poly

You already know how to write the procedure for a square:

```
to square
repeat 4 [fd 50 rt 90]
end
```

To execute the square instruction, you just type `square` at the ? prompt and press return. From now on, I'll write something like this:

```
? square
```

to mean "type `square` at the ? prompt and press return."

When you are playing with designs, it's useful to add variables so that your procedures can have input. Here's how to add a variable for the size of the square:

```
to square2 :size
repeat 4 [fd :size rt 90]
end
```

Here's how you could execute the square2 instruction:

```
? square2 100
```

Try out some numbers of your own. Now suppose you wanted to play with the angle the turtle turns at each corner. We wouldn't have a square any more (unless the input for angle was 90), so let's call the procedure `poly`:

```
to poly :size :angle
repeat 4 [fd :size rt :angle]
end
```

But now we have a problem. Suppose we execute this instruction:

```
? poly 100 60
```

Turns of 60 degrees should make a hexagon (a six-sided polygon), since 360 / 6 = 60. But we only repeat 4 times. So we can add another variable for the number of times to repeat:

```
to poly2 :times :size :angle
repeat :times [fd :size rt :angle]
end
```

Now we can do something like:

```
? poly2 6 100 60
```

Play around with poly2 and see what kind of designs you can make. Remember, Logo can do math for you, so you can do things like:

```
? poly2 7 100 360 / 7
```