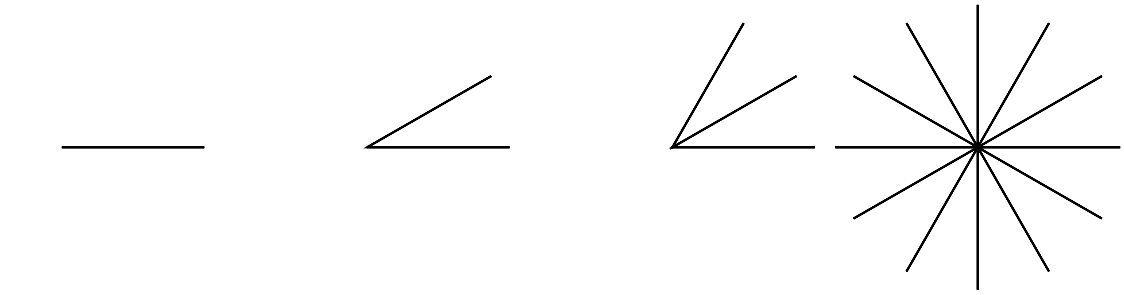
1

Looping



By now you must think that the job of robot programmer is quite tedious. We are sure that you have ideas of nice drawings, but you did not have the heart to draw scripts to draw them. Indeed, the amount of things to type gets larger and larger as the complexity of the drawing augments. In this chapter, you will learn how to reduce the amount of expressions given to a robot using loops. Loops allow you to *repeat a sequence of messages*. With a loop, the script for drawing an hexagon or an octagon is no bigger than the one drawing a square.

1 A Star as a Motivating Example

We would like that a robot draws a star as shown in the picture above. The principle is the following one: A robot has to draw a line, comes back to its previous location, turns from a certain angle and draws another line and so on. The Script **??** makes a robot drawing a line of 70 pixels and coming back to its previous location. Note that in addition, after having drawn the line the robot points in the same direction where it was pointing before drawing the line.

Script 1.1 (*Drawing a line and coming back)*

```
| pica |
pica := Bot new.
pica go: 70.
pica turnLeft: 180.
pica go: 70.
pica turnLeft: 180.
```

Now to draw a star, we have to *repeat* part of the Script **??** and make the robot turns from a given angle, for example 60 degrees. The Script **??** shows how this should be done to obtain a star having 6 branches without using loops.

Script 1.2 (*A star without loop!)*

```
| pica |
pica := Bot new.
pica go: 70.
pica turnLeft: 180.
pica go: 70.
pica turnLeft: 180.
pica turnLeft: 60.
pica go: 70.
pica turnLeft: 180.
pica go: 70.
pica turnLeft: 180.
pica turnLeft: 60.
pica go: 70.
pica turnLeft: 180.
pica go: 70.
pica turnLeft: 180.
pica turnLeft: 60.
pica go: 70.
pica turnLeft: 180.
pica go: 70.
pica turnLeft: 180.
pica turnLeft: 60.
pica go: 70.
pica turnLeft: 180.
pica go: 70.
pica turnLeft: 180.
pica turnLeft: 60.
pica go: 70.
pica turnLeft: 180.
pica go: 70.
pica turnLeft: 180.
pica turnLeft: 60.
```

As you see, this clearly does not scale to have to type all this code that does all the time the same thing. Imagine if we would like to have a star with 60 branches like the star shown in the Script **??**! In fact we would like to be able to repeat a sequence of messages.

Using a Loop. There is a solution to this problem: use a *loop!* There are different kinds of loops. For the moment the loop we present allows you to repeat a given sequence of messages a given number of times. The method timesRepeat: repeats a sequence of messages a given number of times as shown in the Script **??**. The Script **??** defines the same star than the Script **??** but in a much shorter way.

Script 1.3 (*A star with a loop)*

```
| pica |
pica := Bot new.
6 timesRepeat:
    [pica go: 70.
    pica turnLeft: 180.
    pica go: 70.
    pica turnLeft: 180.
    pica turnLeft: 60]
```

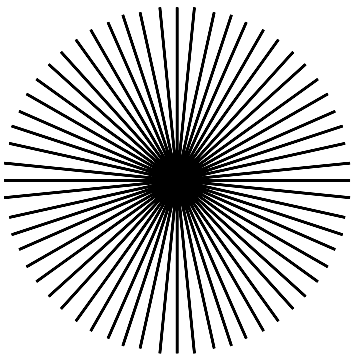
> *n* **timesRepeat: [** *sequence of messages* **]** repeats n times the sequence of messages.

The method timesRepeat: allows you to repeat a sequence of messages a given number of times. In Squeak such a sequence of messages delimited by [and] is called a *block*. Another interesting point is that timesRepeat: is not sent to a robot but to an integer, the number of times the sequence should be repeated. In the Script **??** the message timesRepeat: [...] is sent to 60. Finally note that the number receiving the message timesRepeat: has to be a *whole number* because as in real life there is no sense to do a sequence of messages 0.2785 times.

> The argument of timesRepeat: is a block, *i.e.*, a sequence of messages surrounded by [and]

Type the Script **??** and change the number of times the loop is repeated by replacing 6 by the number you want. Pay attention that 60 should be changed accordingly if you want to generate a complete star. To have a complete star the relation between the angle and the number of repetition should be $angle * n = 360$. In the Script **??** the loop is repeated 60 times and the angle is 6 degrees, so the star is complete.

Experiment 1.1
Write a script that draws A star with 60 branches.



About code indentation. In Smalltalk, the code can be laid out in all kind of ways and the indentation (its shape regardings the left margin) does not change a program. We say it does not affect the sense of the program. However, using a clear indentation really helps the reader to understand the code. We suggest to follow the convention we chose to format timesRepeat: expressions. The idea is that the repeated block of expressions delimited by the characters [and] should form a visual and

textual rectangle. That is why we start the block with [on the next line after the timesRepeat: and align all the expressions inside the block to one tab and finish by] that indicates that the block ends. Note that code formatting is one of the most complex topics because different people like to read their code in different ways. So the one we propose is primarily focused at helping the identification of the repeated messages as illustrated in the following code

2 Exercising Regular Shapes

As you may have noticed, some figures can be obtained by simply repeating sequences of messages, especially the ones produced in Section **??** of Chapter **??** (repeated here as the script **??**).

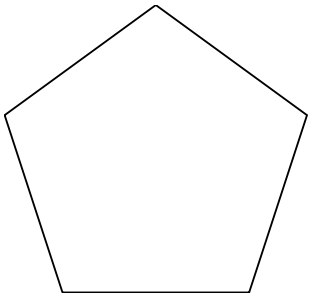
Script 1.4 (*A first square)*

```
| pica |
pica := Bot new.
pica go: 100.
pica turnLeft: 90.
pica go: 100.
pica turnLeft: 90.
pica go: 100.
pica turnLeft: 90.
pica go: 100.
pica turnLeft: 90
```

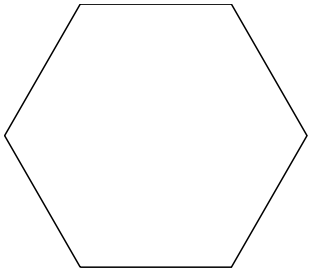
Experiment 1.2
Transform the Script **??** to draw the same square but using the command timesRepeat:.

Now you should be able to draw other regular polygons with a large number of sides.

Experiment 1.3
Draw a pentagon using the method timesRepeat:.



Experiment 1.4
Draw a hexagon using the command timesRepeat:.



If you get the hang of it, try to augment the number of sides of a polygon to a very large number. You may need to reduce the size of the sides so that figure fits within the screen. When the number of

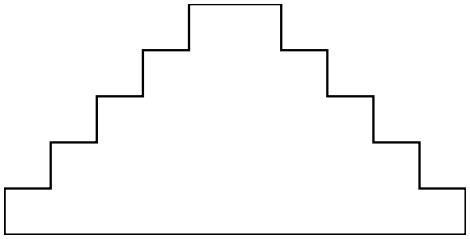
sides is large and the size of the sides is small, the polygon will look like a circle.

3 Pyramids Rediscovered

Remember how you coded the outline of the pyramid of Saqqarah in Experiment **??**? You can simplify your drawing by using a loop as follows:

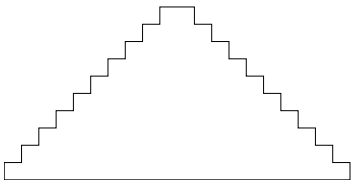
Script 1.5 (*Pyramid script)*

```
| pica |
pica := Bot new.
5 timesRepeat:
    [pica north.
    pica go: 20.
    pica east.
    pica go: 20].
5 timesRepeat:
    [pica go: 20.
    pica south.
    pica go: 20.
    pica east].
pica west.
pica go: 200.
```



Now you should be able to generate pyramids with any number of terraces *with the same number of expressions*, just by changing the numbers of the script.

Experiment 1.5
Try to draw a pyramid with 10 terraces using a variation of Script **??**.

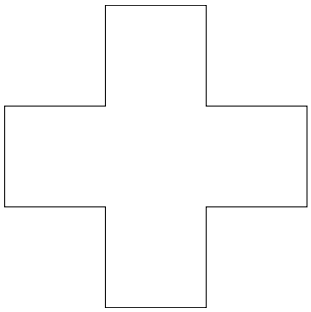


You may want to generate pyramids with an even larger number of terraces. The size of the terraces must be adjusted if you want them to fit within the screen.

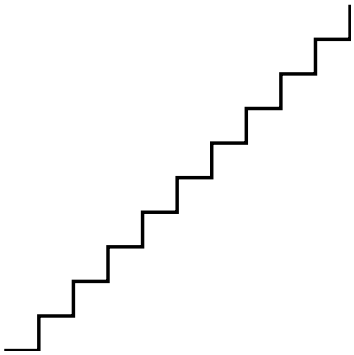
4 Some Selected Problems

As you have seen, the generation of the pyramid involves the repetition of a block of code, which draws two line elements. Once the proper repeating element is identified, one can produce complex picture from elementary drawing, by repeating themselves. The following exercises illustrate this principle.

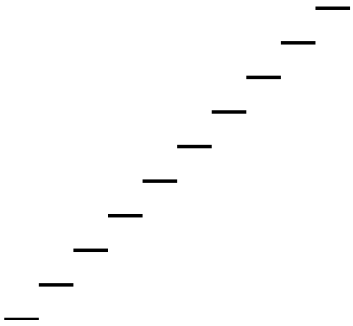
Experiment 1.6 (*Cross)*
Draw the outline of the cross shown on the right using turnLeft: or turnRight: and timesRepeat:.



Experiment 1.7 (*Stair)*
Draw the following stair.

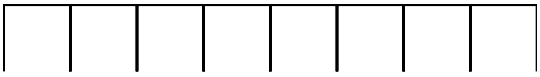


Experiment 1.8 (*Stylized Stair)*
Draw the following stylized stair.

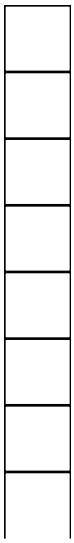


Experiment 1.9 (*A Simple Element)*
Draw the following graphical element.

Experiment 1.10 (*Comb)*
Transform the Script **??** to produce a comb.



Experiment 1.11 (*Ladder)*
Transform the Script **??** to produce a ladder.



Experiment 1.12
Now that you master loops define a loop that draws the tumbling squares of the picture shown at the opening of Chapter **??**.

Summary

Method	Description	Example
n timesRepeat: [a sequence of messages]	repeats *n* times a sequence of messages	10 timesRepeat: [pica go: 10. pica jump: 10]