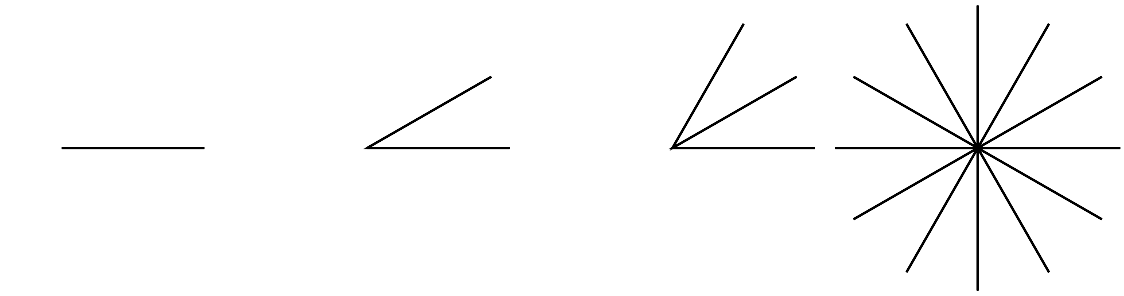
1

Looping



By now you must think that the job of robot programmer is quite tedious. We are sure that you have ideas for nice drawings, but you don't have the heart to write scripts to draw them. Indeed, the amount of things to type gets larger and larger as the complexity of the drawing increases.

In this chapter, you will learn how to reduce the number of expressions given to a robot by using loops. Loops allow you to *repeat a sequence of messages*. With a loop, the script for drawing a hexagon or an octagon is no bigger than the one for drawing a square.

1 A Star as a Motivating Example

We would like a robot to draw a star as shown in the picture above. The principle is as follows: A robot has to draw a line, come back to its previous location, turn a certain angle and draw another line – and so on.

Script 1.1 makes a robot draw a line 70 pixels long and come back to its previous location. Note that in addition, after having drawn the line the robot points in the same direction 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 Script 1.1 – plus make the robot turn a given angle, for example 60 degrees. Turning 60 degrees means that you will get 360 / 60 = 6 branches. Script 1.2 shows how this is 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.
```

As you see, it clearly does not scale to have to repeatedly type all this code that does the same thing each time. Imagine if we wanted to have a star with 60 branches like the star shown in Script 1.1! 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 several kinds of loops. For the moment the loop we present allows you to repeat any sequence of expressions a specific number of times. The method timesRepeat: repeats a sequence of expressions a given number of times as shown in Script 1.3. Script 1.3 defines the same star as Script 1.2 but in a much shorter way. Noticed that the repeated expressions are surrounded by the characters [and].

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 expressions* **]** repeats the sequence of expressions n times.

The method timesRepeat: allows you to repeat a sequence of expressions a specific number of times. In Smalltalk, a sequence of expressions surrounded by the [and] is called a *block*.

timesRepeat: is a message sent to an integer, the number of times the sequence should be repeated. In Script 1.3 the message timesRepeat: [...] is sent to the number 6. Note that there is nothing new here as we already mentioned that even adding two numbers is done by sending message to the first one in Smalltalk.

Finally note that the number receiving the message timesRepeat: has to be a *whole number* because as in real life it makes no sense to do a sequence of expressions 0.2785 times.

The argument of timesRepeat: is a block, that is a sequence of expressions surrounded by [and]. A message argument is the required information needed to make the message execution work (see Chapter **??**). For example [pica go: 70. pica turnLeft: 180. pica go: 70.] is a block, that is a sequence of three expressions: pica go: 70, pica turnLeft: 180 and pica go: 70.

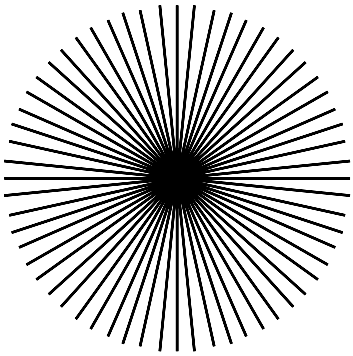
> The argument of timesRepeat: is a block, that is a sequence of expressions surrounded by [and].

Loops at Work. If you compare Script 1.1 with the expressions in the loop of Script 1.3, you will see that there is one extra expression: pica turnLeft: 60. Can you explain why you need such an expression to draw a star? Can you draw a simple relationship between the branch number and the angle that we should turn in addition to draw a branch?

To have a complete star the relation between the angle and the number of repetitions should be $angle * n = 360$. In Script 1.3 the loop is repeated 6 times and the angle is 60 degrees, so the star is complete.

Type Script 1.3 and change the number of times the loop is repeated by replacing 6 by the number you want and generate a complete star.

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 kinds of ways and the indentation (the spacing from the left margin to the beginning of the line) does not change the program's result. We say it does not affect the sense of a program. However, using a clear indentation really helps the reader to understand the code.

We suggest following the convention we used in Script 1.3 to format timesRepeat: expressions and as shown in Figure 1.1. The idea is that the repeated block of expressions surrounded 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:, align all the expressions inside the block with one tab, and finish with the] that indicates where the block ends.

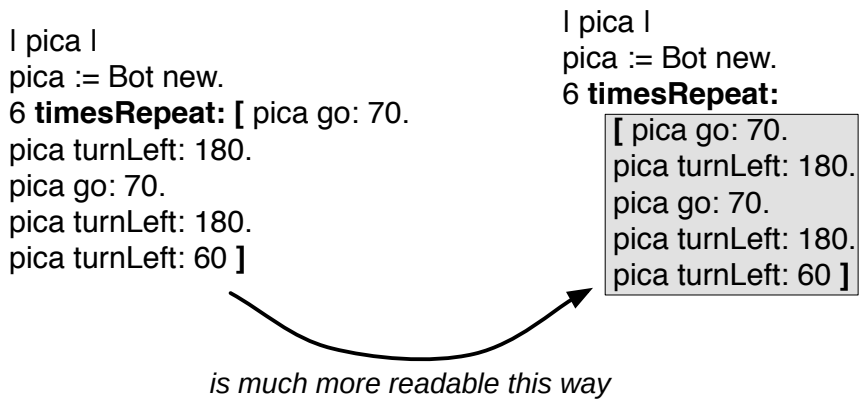


Figure 1.1: The same program not indented and indented using a simple indentation to support loops identification.

Note that code formatting is one of the most complex topics because different people like to read their code in different ways. The one we propose is primarily focused at helping the reader identify the repeated expressions.

2 Exercising Regular Shapes

As you may have noticed, some figures can be obtained by simply repeating sequences of expressions, especially the ones produced in Section **??** of Chapter **??** (repeated here as the script 1.4).

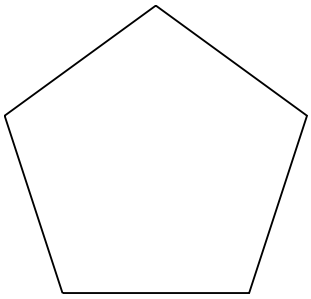
Script 1.4 (*A first square)*

```
| 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
```

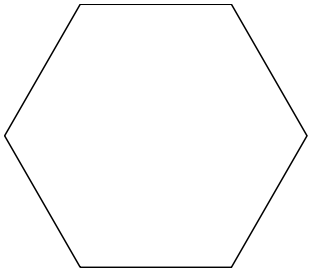
Experiment 1.2
Transform Script 1.4 to draw the same square but using the command timesRepeat:.

Now you can 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:.



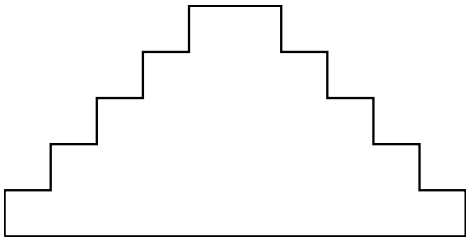
When you get the hang of it, try increasing the number of sides of a polygon to a very large number. You may need to reduce the length of the sides so the figure fits within the screen. When the number of sides is large and the size of the sides is small, the polygon will look like a circle.

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:

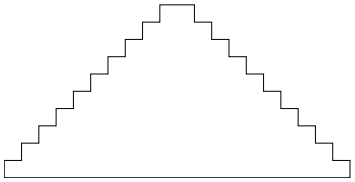
Script 1.5 (*Pyramid script)*

```
| 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 can 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 1.5.

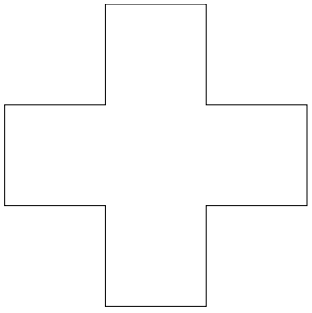


You may want to generate pyramids with an even larger number of terraces. The size of the terraces must be adjusted if you want them to fit within the screen.

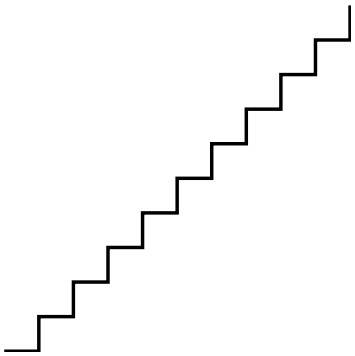
4 Some Selected Problems

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 pictures from elementary drawings, by repetition. The following exercises illustrate this principle.

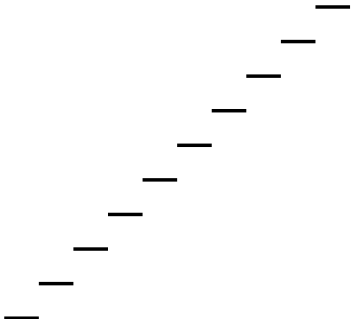
Experiment 1.6 (*Cross)*
Draw the outline of the cross shown on the right using turnLeft: or turnRight: and timesRepeat:.



Experiment 1.7 (*Stair)*
Draw the stairway.

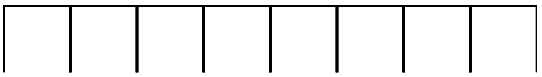


Experiment 1.8 (*Stylized Stair)*
Draw this stylized stairway.

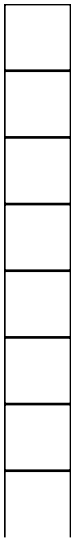


Experiment 1.9 (*A Simple Element)*
Draw the graphical element displayed on the right.

Experiment 1.10 (*Comb)*
Transform Script 1.9 to produce a comb.



Experiment 1.11 (*Ladder)*
Transform Script 1.9 to produce a ladder.



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

Summary

- *n* **timesRepeat: [** *sequence of expressions* **]** repeats the sequence of expressions n times.
- The argument of timesRepeat: is a block, that is a sequence of expressions surrounded by [and].

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