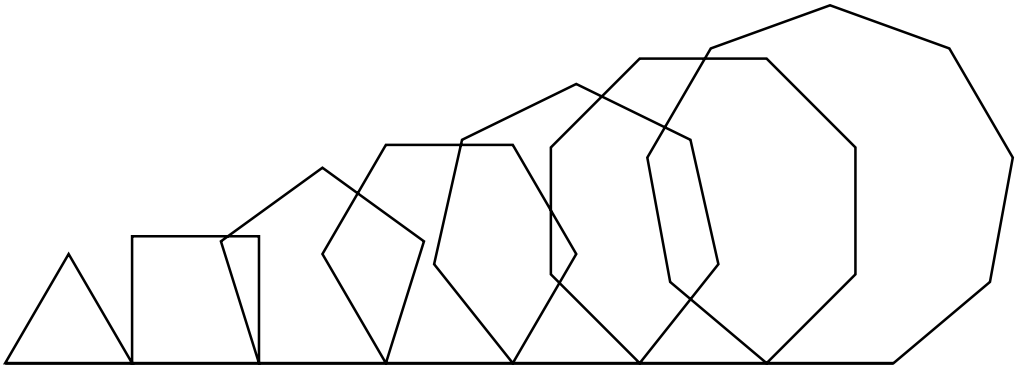
1

Parameters and Arguments



In many previous scripts you sent messages with *arguments*. For example in the message go: 100 you specified that a robot should move a distance of 100 pixels, and 100 is the argument of this message. You learned how to define methods but not how to define methods that require arguments.

In this Chapter, you will learn how to define methods whose behavior can be parameterized, that is whose behavior depends on message argument values. Method parameters act as holes in the method definitions, holes that are filled when the message is sent. First we will define a method with a parameter and invoke it, then we will analyze it.

1 Parameter, you say?!

The method square defined in Chapter **??** is rather limited because the size of the square is fixed once and for all. You probably asked yourself what should be done to draw a square 300 pixels wide, or 175, or 225 or even 23 pixels wide. There is nothing preventing you from defining the methods square300, square175, square225, square23 and so on.

But if you think about it, creating multiple square methods does not solve the problem we have here. We would rather not define a new method each time. We would like to be able to specify the size of the square without having to define a new method for every size! For example, we would like to be able to draw squares whose size is given by a user. For that we wouldn't want to define a method.

What we need is a kind of variable whose value will be assigned when the message is sent, and not before. This kind of variable exists in many programming languages and is called a *parameter*. A

©S. Ducasse

method parameter is a special variable which can take any value you like *at the moment the message is sent*, not when you defined the method.

Come to think of it, this sounds familiar, doesn't it? After all, you know that methods such as go: or turnLeft: take a value (a distance or an angle) at the time the message is sent. Each time you used expressions such as pica turnLeft: 90, pica turnLeft: 32, or daly turnLeft: 65, you defined in the message the value of the angle in the message and the method turnLeft: used this value. In fact this was the *same* method turnLeft: that was executed with a *different value* as value for the angle to turn. So as you see this is really powerful. Now all you have to do is to understand how to define a method able to take arguments at execution time, like the method turnLeft: can.

1.1 The method square:

You have seen that, in Smalltalk, the name of a method is terminated by a colon (:) to indicate that the method requires an argument. So if you want to create a method to draw a square with an arbitrary size, we will use square: as its name. You define the method square: as shown in Method 1.1. This method is then used in Script 1.1.

Method 1.1

```
square: size
  "Draw a square of the given size"

  4 timesRepeat:
            [ self go: size.
            self turnLeft: 90 ]
```

Script 1.1 (*Using the method square:)*

```
| pica |
pica := Bot new.
pica square: 10.
pica go: 300.
pica square: 20
```

Now let's analyze definition of method square: (method 1.1). To define a method that requires one argument, you terminate the method name with a colon : and follow it with the name of the parameter, which is size in Method 1.1.

The parameter just represents a variable whose value is defined when the message is sent (not when the method is defined). In the script 1.1, during the first message square: 10 size will have the value 10. Then during the message square: 20, size will have the value 20. The argument, however, does not need to be explicitly declared using vertical bars | and |.

In the method 1.1 the name of the parameter is size. The name of the parameter should represent what it is used for. Note that we could have named this parameter length as in method 1.2, or distance — as long as we replace all the occurrences of size in the method body with length. Note that method 1.2 and method 1.1 give *exactly* the same results.

Method 1.2

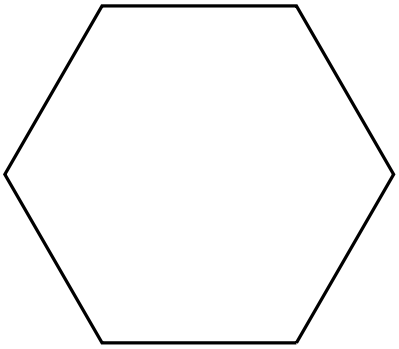
```
square: length
  "Draw a square of the given size"

  4 timesRepeat:
        [ self go: length.
        self turnLeft: 90 ]
```

2 Practicing

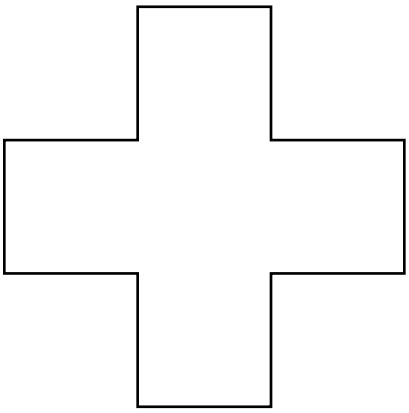
Now this is time to practice a bit. Let's start with a simple exercise.

Experiment 1.1 (*hexagon:*)
Define the method hexagon: that draws a hexagon with sides of the length passed as argument.



Experiment 1.2 (*Cross)*
Transform the script given below into a method named cross: that draws a cross of the size that is passed as argument. You should then be able to execute the expression: pica cross: 100. Hint: notice that 100 / 2 = 50.

```
|pica |
pica := Bot new.
4 timesRepeat:
        [ pica go: 50.
        pica turnLeft: 90.
        pica go: 100.
        pica turnRight: 90.
        pica go: 100.
        pica turnRight: 90.
        pica go: 50 ]
```



©S. Ducasse

3 Experimenting with Multiple Arguments

Of course, it would be better to have a method drawing a polygon with a given number of sides *and* size. The question is how do we create a method having two arguments? You can create a method with two arguments by writing a method name with two colons and placing one argument name after each colon.

> To define a method with multiple arguments, terminate each word in the method name with a colon, and place each parameter after its corresponding word in the method name.
>
> The method named polygon:size: requires two arguments. The definition of the method polygon: numberOfSides size: size defines two arguments, numberOfSides and size.

Its code is shown below. You can then simply send the message pica polygon: 7 size: 100.

Method 1.3

```
polygon: numberOfSides size: size
   "Draws a polygon with the given number of sides and size"

   | angle length |
   angle := 360 / numberOfSides.
   length := 4 * size / numberOfSides.
   sideNumber timesRepeat:
                        [ self go: length.
                        self turnLeft: angle ]
```

Implementation Remark. You may wonder why we defined the length as 4 * size / numberOfSides. We decided to make the polygon's perimeter equal to the perimeter of a square (4 * size is the perimeter of a square. So dividing it by the number of sides and asking a robot to walk that distance the number of sides, makes the robot walking the square perimeter.) that has a side of length size. This way the perimeter is constant regardless of the numberOfSides, and all the polygons will be displayed using about the same fraction of the screen.

Here is the code of the method polygon100: which draws a polygon with the given number of sides, each side having a length of 100 pixels.

Method 1.4

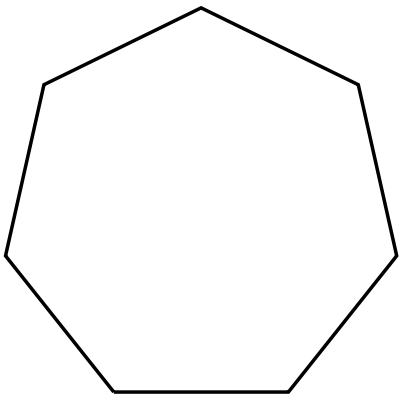
```
polygon100: numberOfSides
   "Draws a polygon with the given number of sides; the length of each
   side is 100 pixels"

   | angle |
   angle := 360 / numberOfSides.
   sideNumber timesRepeat:
                        [ self go: 100.
                        self turnLeft: angle ]
```

This method has one argument, numberOfSides, and one variable, angle. Both of them are used within the code of the method. numberOfSides's value is specified by the message that uses the method, for example pica polygon100: 7. The variable angle is initialized by computing the angle corresponding to the parameter values (namely 360 / 7 in our example). For any value of numberOfSides, angle will get the right value for the polygon.

Script 1.2 (*Using the method polygon100:*)

```
| pica |
pica := Bot new.
pica polygon100: 7.
```



You may think that the name of the parameter numberOfSides is long. However, this name can easily be understood by any person reading the method. As we already discussed in Chapter **??**, it is quite important that anyone be able to read your code — almost like a story.

Experiment 1.3
Define the method rectangleWidth:height: which draw a rectangle with the given width and height.

Experiment 1.4
By slightly modifying the method cross:, define the method crossWidth:height: that can draw the crosses shown in Figure 1.1. Note that a normal cross corresponds to pica crossWidth: 30 height: 60

©S. Ducasse

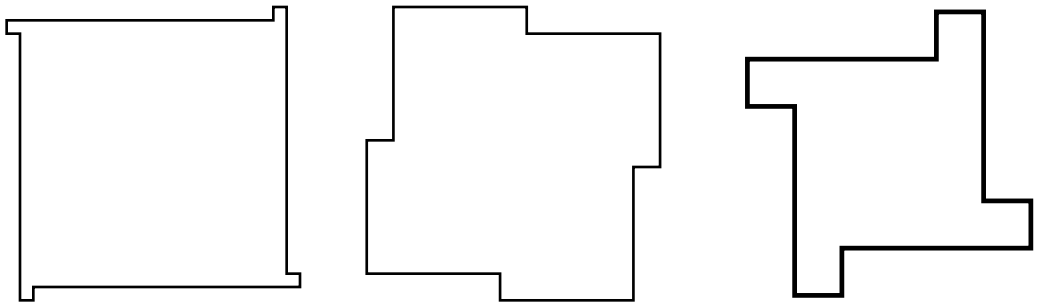


Figure 1.1: Three crosses produced by crossWidth:height: messages. The cross on the left is from pica crossWidth: 5 height: 50; in the middle, from pica crossWidth: 50 height: 5; on the right, from pica crossWidth: 10 height: 20.

4 Parameters and Variables

Now that you've practiced a bit, it is time to look more carefully at the difference between variables and parameters. Let's compare Script 1.3 and Method 1.5 that I defined earlier (see copies below).

In Script 1.3, first the variable size is declared (line 1), then a value is assigned to it (line 3) and finally it is used as the argument of the method go: (line 5).

Script 1.3 (*The square script using a variable)*

```
(1)  | pica size |
(2)  pica := Bot new.
(3)  size := 10.
(4)  4 timesRepeat:
(5)          [ pica go: size.
(6)          pica turnLeft: 90 ]
```

In method 1.5 there are examples of two features of parameters. First, the parameter size is declared because it appears after a colon in the method name (line 1). Second, it is used as the parameter of the message go: (line 5). A parameter does not have to be initialized because it always gets the value specified in the message that invokes the method. For example when the message pica square: 20 is sent to pica, then the parameter size of method 1.5 gets as value 20.

Method 1.5

```
(1) square: size
(2)  "Draw a square of  given size"
(3)
(4)  4 timesRepeat:
(5)              [ self go: size.
(6)              self turnLeft: 90 ]
```

No need to declare parameter. Unlike other variables, a parameter doesn't have a variable declaration between vertical bars ||. A parameter is declared when it appears after a colon : in the first line

of the method's definition.

Parameter cannot be assigned. Another difference is that parameters cannot be modified the way other variables can. You cannot assign new values to parameters inside method bodies. For example, in method 1.5 the expression size := 100 is impossible.

Variable initialization. The other difference is the way values are assigned to variables and parameters. A variable value is changed using :=. A parameter value is initialized when the method is executed. For example, pica square: 10 initializes the parameter size with the value 10. A parameter is a variable. However, the parameter's value is only known when a message is sent and the method executed.

Besides those three differences, a parameter is used in the code of the method like any other variable.

5 Arguments and Parameters

I used the two terms *arguments* and *parameters* for related but different ideas. An argument is the specific object passed in a message. A parameter is the variable used in a method definition, whose precise value isn't known when the method is defined[1].

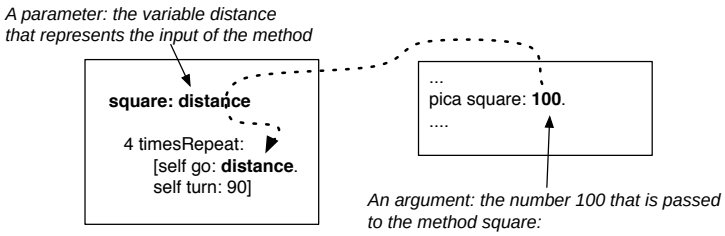


Figure 1.2: The connection between an argument (an object) and a parameter (a variable).

In Figure 1.2, in the message square: 100, the number 100 is the message argument. When the method square: is executed its parameter distance is initialized to 100, the value of the argument.

Another way to understand the difference between an argument and a parameter is that a parameter is a variable inside a method that represents an input to the method, while an argument is the actual value you pass to this input as shown in Figure 1.3.

Note that a parameter can also be used as an argument in other message sends. For example in the definition of the method square: (method 1.1), the parameter size is used as the argument in the message go: size.

A message argument can also be a variable. For example in script 1.4, the argument of the first message square: is the value of the variable dist, that is 100. The argument of the second message square: is the value of the expression dist + 200, that is 300. The parameter size of the method square: gets the value from the first message, and then the value 300 from the last message.

[1]Many authors define these terms differently. Some use "actual parameters" for what we call "arguments", and "formal parameters" for what we call "parameters". Others use the terms "parameter" and "argument" interchangeably.

©S. Ducasse

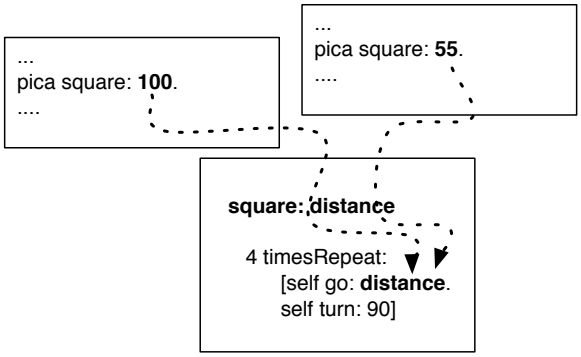


Figure 1.3: The value of the argument is bound to the parameter during the method execution.

Script 1.4 (*A variable as argument)*

```
| pica dist |
pica := Bot new.
dist := 100.
pica square: dist.
pica go: 300.
pica square: dist + 200
```

About Method Execution. In a first reading you can simply skip this paragraph since it goes into details that you do not need to know at first. I wrote it because I wanted to answer the questions of the most curious readers but I could have removed this paragraph without problem. When a method is executed new variables are created. These variables are the message receiver, self, and the method parameters (that refer to the method arguments) such as size in Figure 1.4. Figure 1.4 shows the effect of sending the message square: length to a robot referred to by the variable pica, when the variable length references the number 100.

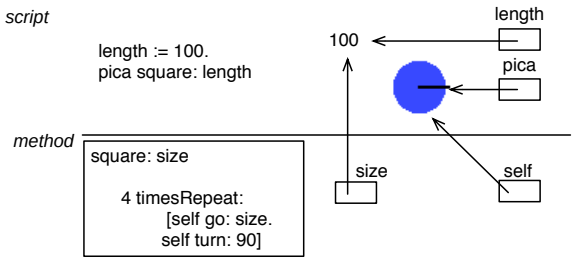


Figure 1.4: When a method is executed, new variables are created that refer to the arguments and the receiver.

When the method square: is executed, the variable self refers to the message receiver, that is the robot pointed to by the variable pica, and the parameter size refers to the value of the variable length,

that is the number 100. The same process occurs for each message send. For example, the execution of the expression daly square: 200 assigns to self the robot referenced by the variable daly, and assigns to size the number 200.

This may look complex but you do not have to worry about it. These are the hidden steps Squeak takes to make sure that parameters are initialized with the values in the messages.

Summary

1. A method parameter is declared right after the colon indicating the position of the parameter. It must not be declared as a variable.

2. To define a method with multiple arguments, terminate each word in the method name with a colon, and place each parameter after its corresponding word in the method name.
   The method named polygon:size: requires two arguments. The definition of the method polygon: numberOfSides size: size defines two arguments, numberOfSides and size.

©S. Ducasse