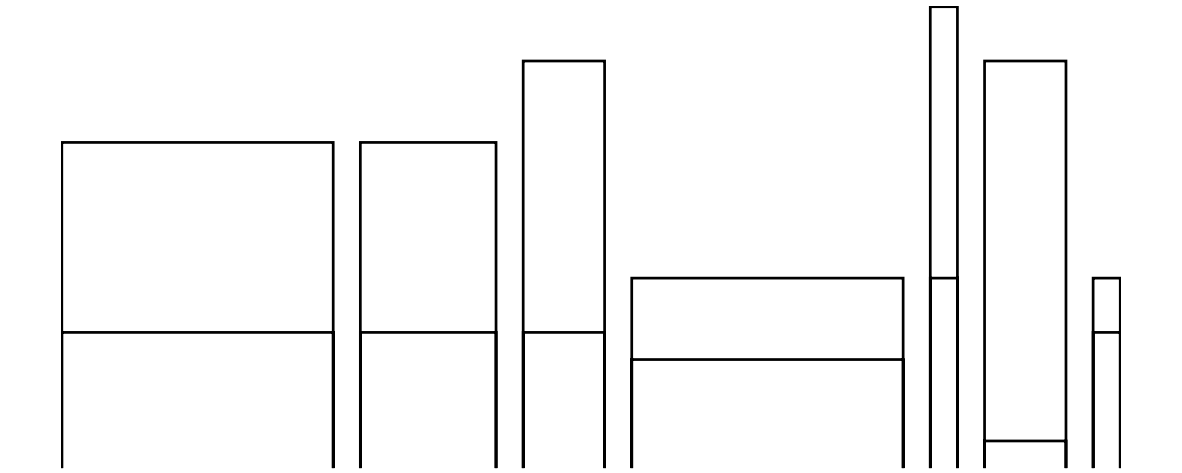
1

Variables



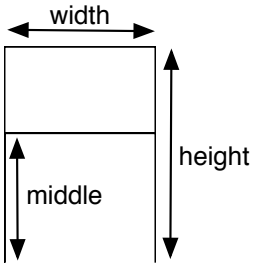
We are constantly giving names to human beings or things. For example, we give names to people, to dogs or to cars. By doing that we are *associating* something to a word or a symbol. Once this association is done, we then later use the word or symbol to *refer to* or to interact with the thing associated with it. Sometimes, names are for a lifetime or sometimes just for a short period of time. Sometimes names refer to other names. For example, an actor has several names, a public name, a civil name and a name of the character he is playing in a movie. In a programming language, we also need to be able to name things and variables are used for that.

In this chapter, we introduce *variables* which are placeholders for objects and show how variables help to simplify programs. Moreover using variables is sometimes necessary for programming. Finally, as the complexity of problems you will facing will increase, you will see that you will need to express dependencies between variables. For example, the width of a rectangle can be 2/3 of its length. In this chapter we show how to use variables to express dependencies between numbers.

1 A World of As

As asked in Chapter **??**, suppose that we want to use a robot to write letters. The character A is characterized by a *height*, a *width* and a *middle* from which the middle line of the A should be drawn as shown by Figure 1.1.

Experiment 1.1
Propose a script that draws a character A of 100 pixel height, 70 pixel width and 60 pixel middle (distance between the ground and the first horizontal line).

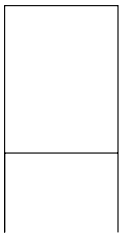


Variations on A. The script you wrote for answering the Experiment 1.1 should look somehow like the following script 1.1.

Script 1.1 (*An A of 100)*

```
| pica |
pica := Bot new.
pica north.
pica go: 100.
pica east.
pica go: 70.
pica south.
pica go: 100.
pica west.
pica jump: 70.
pica north.
pica go: 60.
pica east.
pica go: 70
```

Experiment 1.2 (*frAnkenstein)*
Change the Script 1.1 to draw an A of 200 pixel height, 100 pixel width and 70 pixel middle.



As the previous experiment 1.2 shows you, every times we want to produce an A of a different size we have to change *everywhere* and *synchronously* the numbers that represent the height, the width and the middle of the letter. By synchronously, we mean that 100 should become 200, 70 should become 100 and 60 should become 70 without mixing them.

Experiment 1.3
Change the Script 1.1 to draw other A letters with different sizes of your choice. Try to reproduce some of the As that compose the opening picture of this chapter.

As you may notice, having to change the values everywhere is tedious. Moreover, you also may risk becoming confused and forget to change a value at one place. This can result in breaking the script. You can easily imagine that in complex programs this is a real problem to change values in such a way.

2 Variables to the Rescue

As seen with the previous exercises producing letters of different size can be tiresome and errorprone as you can mixed the number representing the various lengths of a letter. We have to take care to change all the values everywhere. In fact, we would like to be able to:

- *declare* the height, width and middle of an A once for all,
- being able to *refer to* these values, and
- possibly *change* the values if necessary.

This is exactly what a variable allows us to do! Amazing isn't it? A variable is a *name* to which we *associate a value*. We must *declare* it and *associate* a new value to it. Then we can *refer to* a variable and obtain the *value* associated with this variable. It is also possible to *modify* the value associated with a variable and associate it a new value. The variable value can be a number, a collection of objects, ... even a robot. We now illustrate how to declare, associate a value and use a variable.

> A variable is a *name* to which we *associate a value*. We *declare* a variable and *associate* a value to it. Then we can *refer to* a variable and obtain its *value*. It is also possible to *modify* the value associated with a variable and associate a new value to it.

Declaring a variable

Before using a variable, we have to *declare* it, that is telling to the system the name of the variable that we want to use. We declare variables by enclosing them with vertical bars || as shown by the following example which declares three variables height, width and middle. To be exact, vertical bars || declare temporary variables that is variables that only exist during the execution of the script.

```
| height width middle |
```

Assigning a value to a variable

Before using a variable it is better to give it a value. Associating a value is called *assigning* a value to a variable. In Smalltalk, := is used to assign a value to a variable. In the following script after declaring the three variables we assign 100 to the variable height, 70 to the variable width and 60 to the variable middle. When this is the first time that we assign a value to a variable, we say that we *initialize* it.

```
| height width middle |
height := 100.
width := 70.
middle := 60
```

:= assigns a value to a variable. Example: height := 120 assigns the 120 to the variable height. length := 120 + 30 assigns the result of the expression 120 + 30 that is 150 to the variable length.

When this is the first time that we assign a value to a variable we say that we initialize it.

Referring to Variables

To refer to the value assigned to a variable – we also say use a variable, you simply write its name in a script. In the following script, after being declared line 1, the variable height is initialized with the value 100 in line 2 and used line 4 to say to the created robot to go forward from the height variable's value, here 100.

```
| pica height |
pica := Bot new.
height := 100
pica north.
pica go: height
```

Generally a variable must be *declared* and *initialized*, before being used.

About pica

Yes, you guessed right! pica is also a variable, just a variable whose value is a robot. Hence, | pica | declares that you use a variable named pica. The expression pica := Bot new initializes the variable with a value, here a new robot. Then we use this robot by sending messages to it via the variable pica, for example pica go: 100.

3 Using Variables

Now let us explore the benefits of using variables and show you some powerful properties. In particular, we show that been able to express relationships between variables is really powerful.

Once we introduce variables in the Script 1.1, we obtain the Script 1.2.

Script 1.2 (*An A of 100)*

```
| pica height width middle |
pica := Bot new.
height := 100.                "initializes the variables"
width := 70.
middle := 60.
pica north.
pica go: height.              "then we use the variables"
pica east.
pica go: width.
pica south.
pica go: height.
pica west.
pica jump: width.
pica north.
pica go: middle.
pica east.
pica go: width
```

Now changing variable values is easier. Change some values to convince yourself. You should be able to draw all the As that are drawn in the first picture of this chapter. Now to produce a new letter with different lengths you *just* have to change their values during their initialization as shown in script 1.3 and whose result is presented in Figure 1.1.

Script 1.3 (*An A of 100)*

```
| pica height width middle |
pica := Bot new.
height := 30.          "initializes the variables"
width := 200.
middle := 10.
...
```

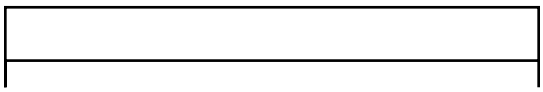


Figure 1.1: A flat A letter simply created with **height = 30**, **width = 200**, and **middle = 10**.

Using variables you can easily create a lot a different characters and solve a lot of problems in the future. It is now worth to step back and look at the power provided by variables.

Analysing the Power of Variables. These experiments illustrate the power of using variables. Variables let you name a thing, being a robot, a length, or anything else. Then you can use these names instead of repeating the values that you associated with the variables. Using variables make your scripts more easy to change as you can simply change the variable values. In addition a variable can

be holding any kind of values: up until now you assigned robots or numbers to variables but it could be a color (for example yellowCol := Color yellow), a sound or any Squeak objects.

Note also that by using variables your scripts are much more readable and easier to understand. To convince yourself just compare the scripts 1.1 and 1.2. The simple fact of using variables with names such as “width” and “height” helps you to understand how the letter is drawn.

3.1 Expressing Relationships between Variables

The characters in a font should respect certain proportions to ease reading, that is the lengths that describe a character are not freely chosen but maintain certain proportions between them.

In our simple character A, let us decide that the width should be 2/3 of the height and that the middle should be at 3/5 of the height. We can express these relationships using variables as shown in the following Script 1.4. Indeed the value of a variable does not have to be limited to a simple number but can be the result of a complex computation.

Script 1.4 (*Towards a Solution)*

```
| pica height width middle |
pica := Bot new.
height := 120.
width := 120 * 2 / 3.
middle := 120 * 3 / 5.
...
```

If you analyze a bit the Script 1.4, you realize that it is not optimal. The relationships are expressed between the variables, but still the value 120 (in lines 3,4 and 5) has to be changed manually when we want to produce a different A holding the same relations. The solution is to use the variable height instead of 120 as shown in Script 1.5. In this script, the value of the variables width and middle depends on the one of height. The value of a variable can be expressed by using other variables. The expression width := height * 2 / 3 expresses that the width of the character is equal to 2/3 of the height.

Script 1.5 (*Making width and middle depending on height)*

```
| pica height width middle |
pica := Bot new.
height := 120.
width := height * 2 / 3.
middle := height * 3 / 5.
pica north.
...
```

Important. The only constraint we have while expressing relationships between variables is that the value of variable used in the definition of another should be known. For example, in Script 1.5, the value of height is known while computing the value of width and middle. On the contrary, in Script 1.6 the value of height is not known when the value of width is computed, this leads to an error. We will elaborate more on errors in Chapter **??**.

Script 1.6 (*Problematic width initialization)*

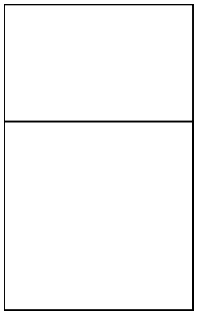
```
| height width middle |
width := height * 2 / 3.
height := 120.
middle := height * 3 / 5.
```

Now you see that variables are really useful to express complex relationships and can hold any kind of objects.

4 Some Experimentations

We recommend you gain some experience with the following problems.

Experiment 1.4 (*Golden Rectangle)*
A golden rectangle is a rectangle whose height is 1.6 times its width — 1.6 is an approximation of the golden number $\frac{1+\sqrt{5}}{2}$ which is the side of a regular decagon inside a circle of unity. The nice property of such a rectangle is that if we draw inside the rectangle a square with size the rectangle's width, then the left part of the rectangle is again a golden rectangle. Such a property is then infinite. Ancient architects were using a lot such a golden proportion in their constructions. Every rectangle created this way is a golden rectangle. Define a script that draws a golden rectangle. $\frac{1+\sqrt{5}}{2}$ is expressed in Smalltalk as `1 + 5 sqrt / 2`.



Experiment 1.5
Explain why none of the following scripts do not draw an A of 120 pixel height.

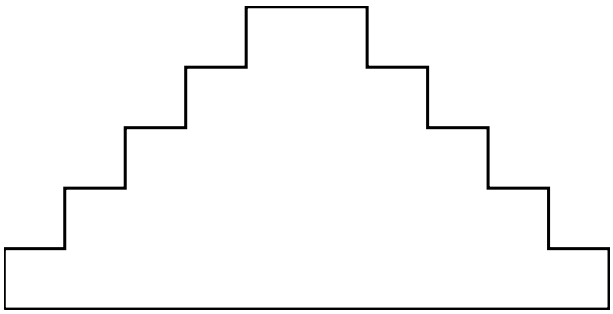
```
| pica height |
pica := Bot new.
height := 120.
pica north.
pica go: 100.
pica east.
pica go: 70.
pica south.
pica go: 100.
pica west.
pica jump: 70.
pica north.
pica jump: 50.
pica east.
pica go: 70
```

```
| pica height |
pica := Bot new.
pica north.
pica go: height.
pica east.
pica go: 70.
pica south.
pica go: height.
pica west.
pica jump: 70.
pica north.
pica jump: 50.
pica east.
pica go: 70
```

Pyramids rediscovered. In Section **??** of Chapter **??** in the Script **??** we defined the outline of the pyramid of Saqqarah the following way.

Script 1.7 (*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.
```



Experiment 1.6
Introduce the variable terracesNumber that represents the number of terraces that the pyramid can have.

Experiment 1.7
In the previous script introduce the variable terraceSize that represents the size of a terrace.

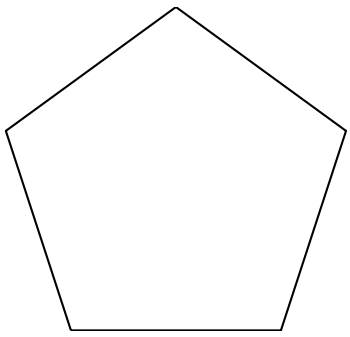
5 Another Example of Variable Use

The use of variables greatly simplifies the definition of scripts where some of the *variables* depend on other ones. In this section, we shall see how the use of variables gives great leverage when dealing with loops. The Chapter **??** will go deeper in showing that the combination of variables and loops is powerful.

Let us look again for a moment at the experiements **??** and **??** in which a Bot was asked to draw a pentagon and a hexagon.

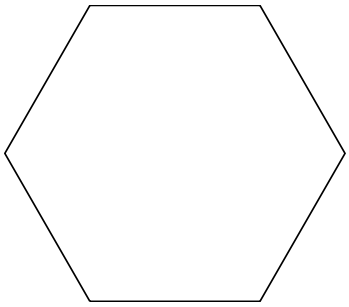
Script 1.8 (*Pentagon)*

```
| pica |
pica := Bot new.
5 timesRepeat:
        [ pica go: 100.
        pica turnLeft: 72 ]
```



Script 1.9 (*Hexagon)*

```
| pica |
pica := Bot new.
6 timesRepeat:
        [ pica go: 100.
        pica turnLeft: 60 ]
```



The difference between the two scripts is that one must change the number of sides (let us call it s) *and* the magnitude of the turn (let us call it T) such that the product $s \times T$ is equal to 360. Wouldn't it be nice if we could write a script, in which we only would have to change a single number, let us say the number of sides since this is the easiest parameter to choose? This is possible by introducing variables. Try to come up with your own solution.

Here is a script where this is done (Script 1.10). Try it before we discuss more about it.

Script 1.10 (*Regular polygon)*

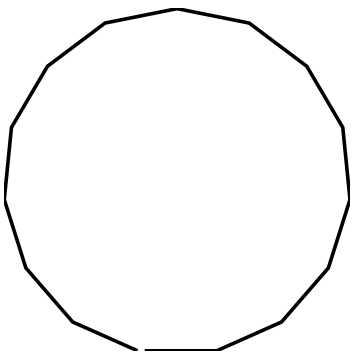
```
| pica sides angle |
pica := Bot new.
sides := 6.
angle := 360 / sides.
sides timesRepeat:
        [ pica go: 100.
        pica turnLeft: angle ]
```

We have introduced two new variables — sides and angle — used to hold the needed values. Then, the line sides := 6 assigns the value 6 to the variable sides and the line angle := 360 / sides assigns a value to the variable angle, which is the result of 360 divided by the value held in the variable number. The value of the variable angle is then used as argument of the command turnLeft: given to the robot in the repeating block.

6 Regular Polygons with Fixed Sizes

If you have used Script 1.10 with a large number of sides, you will have noticed that the resulting figure did not fit on the screen. The next exercise asks you to fix this by reducing the length of the sides when the number of sides augments.

Experiment 1.8
Modify Script 1.10 so that the size of the regular polygon stays constant as the number of sides changes.
Hint: introduce a variable length which is set to a fixed length divided by the number of sides.



Summary

1. A variable is a *name* to which we *associate a value*. We must *declare* it and *associate* a value to it. Then we can *refer to* a variable and obtain the *value* associated with this variable. It is also possible to *modify* the value associated with a variable and associate a new value to it.

2. A variable can be used at any place where its value can be used.

3. When this is the first time that we assign a value to a variable, we say that we *initialize* it.

4. := assigns a value to a variable. Example: height := 120 assigns the 120 to the variable height. length := 120 + 30 assigns the result of the expression 120 + 30 that is 150 to the variable length.

5. A variable must be *declared* and *initialized*, before being used.

Expressions	Description	Example
| ⟨variable name⟩ |	Declaration of a variable	| pica height |
⟨length⟩ := ⟨ *expression* ⟩	Assigns the value of expression to a variable	length := 40 length := 30 + 20
	Uses a variable's value	pica go: length
	Uses and changes the value of a variable	length := length + 10