1

Strings and Tools to Understand Programs

In this chapter I present an important concept: the notion of strings. A string is a sequence of characters that represents words or sentences. Strings are used to communicate with users. We shall use strings in the subsequent chapters as a tool to help you understand condition and conditional loops. In this chapter I limit myself to the most important aspects of strings and present the minimal information that you will use in subsequent chapters. I also present how you can use strings to understand how programs are executed. Of course, I suggest you try to also use the debugger to understand the experiments I propose.

1 Strings

Strings are used to represent information and present it to the user. Strings are delimited by single quotes (') and they can contain white space characters. For example, the following string 'squeak is cool' represents a sequence of 14 characters: s q u e a .... Note that a space in also a character. A string can contain any number of characters, even zero. " is an empty string. 'a' is a string with only the character a. ' ' is a string with only the character space.

A string is a sequence of characters delimited by single quotes '. A string represent textual information such as words or sentences and is used to display information to the user.

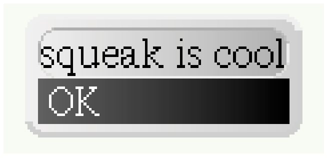
Selecting a string and printing it (menu print it) prints the same string. Several methods are defined on strings: the most important one in the context of this book is the method , that given a string as receiver and one string as argument returns the concatenation of the two. It is for example possible to replace a string into another one using the method copyReplaceAll: as shown below.

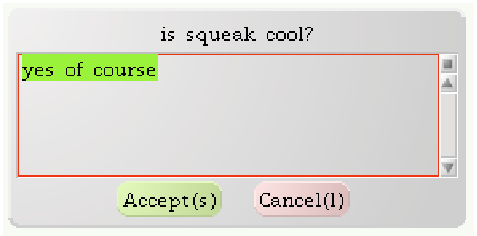
```
'squeak'                                     "the value of a string is itself"
—Printing the returned value:   'squeak'

'a'"a string can be composed of only one character"

''                                    "an empty string"

'squeak' , 'is cool'                          "concatening two strings"
—Printing the returned value:   'squeakis cool'

'', 'squeak', ' ', 'is cool'                  "concatening multiple strings"
—Printing the returned value:   'squeak is cool'

'Squeak is not cool' copyReplaceAll: 'not' with: 'really'
—Printing the returned value:   'Squeak is really cool'
```

2 Communicating with the User

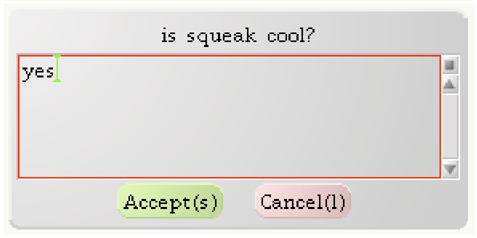
Squeak offers some tools to display and request information. The PopUpMenu allows one to bring up a menu and to display some sentences. For example, the expression PopUpMenu inform: 'squeak is cool' pop ups a small window displaying the message 'squeak is cool' and waits that the user presses the ok button. FillInTheBlank allows you to request some input from the user. For example the expression FillInTheBlank request: 'is squeak cool?' prompts a dialog box with an input field and waits that the user fills the input field and presses the accept button or simply the cancel button. The result of this expression is a string that represents the contents typed by the user. We can also specify a contents by default using the message request:initialAnswer: as shown in the following script.

```
PopUpMenu inform: 'squeak is cool'
```





```
FillInTheBlank request: 'is squeak cool?'
—Printing the returned value:   'yes'

FillInTheBlank request: 'squeak cool?' initialAnswer: 'yes of course'
—Printing the returned value:   'yes'
```



3 Strings and Characters

A string is composed of characters. Individual characters are prefixed by a dollar sign $. For example, $a is character representing the letter a. Note that while individual characters are prefixed by the dollar sign $, when you edit a string we simply type characters without the dollar sign.

Several methods allows one to access characters of strings. For example, the method first, second, third return the first, second, and third characters of a strings. size returns the string character number, at: aNumber returns the character at the specified place, at: aNumber put: aCharacter replaces the character at the position by a new character. The method copyUpTo: aCharacter returns the beginning of a string up to the first character equals to aCharacter.

```
'squeak is cool' first
—Printing the returned value:   $s

'squeak is cool' size
—Printing the returned value:   14

'squeak' at: 5
—Printing the returned value:   $a

'squeak is cool' at: 11 put: $f
—Printing the returned value:   'squeak is fool'

'squeakiscool' copyUpTo: $i
—Printing the returned value:   'squeak'

'squeak is cool' copyUpTo: Character space
—Printing the returned value:   'squeak'
```

To create character that do not have a graphical representation such as space, tab, carriage return, we simply send a message to the class Character. The messages Character space, Character tab, and Character cr return respectively the space, tab and carriage return character.

The following script 1.1 shows how we insert a carriage return inside a string. Note that the method at:put: does not return the modified string but the character that was inserted. This is an example where the effect of the message and its results are clearly different. Printing the result of the message 'squeak is cool' at: 7 put: Character cr does not illustrate the effect of the method. Therefore we print the modified string.

Script 1.1 (*Inserting a carriage return)*

```
|string|
string := 'squeak is cool'.
string at: 7 put: Character cr.
string —Printing the returned value:   'squeak
is cool'
```

A character can also be converted into a string by sending it the message asString.

Script 1.2 ()

```
'sque', $a asString, 'k'
—Printing the returned value:    'squeak'
```

4 Strings and Numbers

A string can represent a number, for example the *string* '10' is a textual representation of the number 10. However, a string is not a number. A string does not know out to perform any mathematical operation and a number does not know how to behave as a string. For example we cannot concatenate two numbers or add two strings. However, a number knows how to produce a string which represents it using the method asString. In addition a string knows how to how to convert a representation of a number into a number using the method asNumber.

> A string can represent a number but this is not a number. For example, the string '69' is composed of the two characters: $6 and $9. To obtain the string representing a number, send the message asString to it.

There is a difference between the number 10 and the string '10'. 10 represents the mathematical number 10 while the string '10' represents the number 10. The string '10' is composed of two characters: $1 and $0. The string '10' is composed of two character $1 followed by $2.

```
10 , 12
-> error! a number does not know the message ,

'10', '12'
—Printing the returned value:    '1012'

10 asString
—Printing the returned value:    '10'

10 asString , 12 asString
—Printing the returned value:    '1012'

'10' asNumber
—Printing the returned value:    10
```

5 Using the Transcript

Squeak offers several powerful tools to understand program execution such as the debugger (see Chapter **??**). Another tool is call the Transcript. A transcript is a window in which you can display information given as strings. To open a transcript, drag into the desktop the thumbnail that is available in one of the flaps or you choose the **transcript** item of the **open...** menu entry. This opens a window as shown in Figure 1.1.

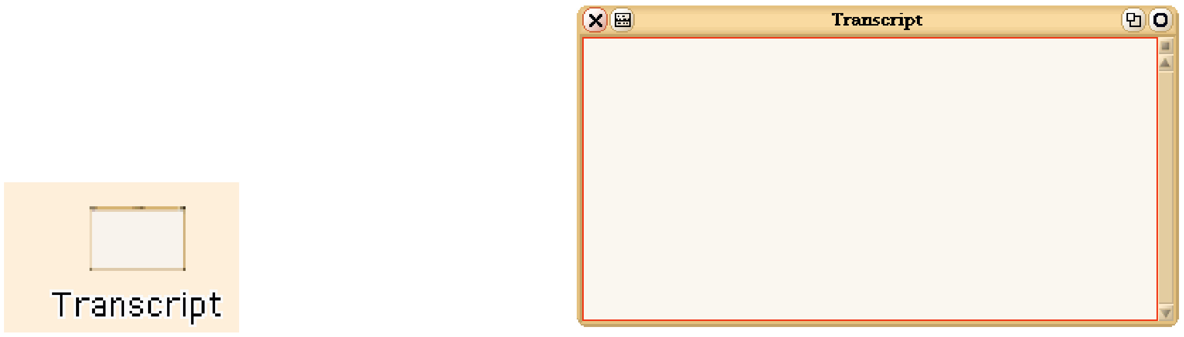
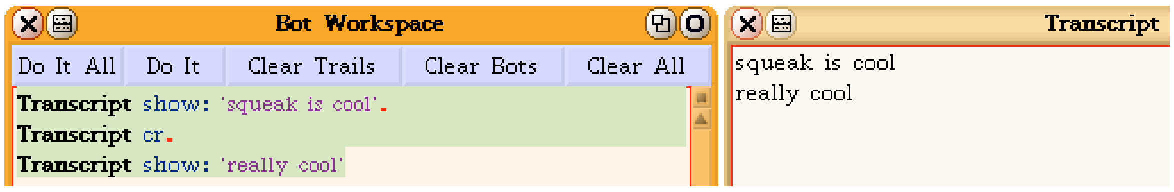


Figure 1.1: To bring a transcript drag and drop the thumbnail that you can find in a flap.

To display information on the transcript there are two main messages show: and cr. The message show: aString displays the string in the transcript and the message cr inserts a new line in the transcript.

Script 1.3 (*Displaying information into a Transcript)*

```
Transcript show: ’squeak is cool’.
Transcript cr.
Transcript show: ’really cool’
```



```
Transcript show: 'squeak is cool'.
Transcript cr.
Transcript show: 'really cool'
```

```
squeak is cool
really cool
```

Figure 1.2: Writing to the transcript.

Note that the Transcript only displays strings. Therefore when we want to display a number we have to obtain a string representing it using the method asString as shown by script 1.4.

Script 1.4 (*Converting a number in a string before displaying it)*

```
Transcript show: ’21 + 21 is: ’, 42 asString ; cr

| answerNumber |
answerNumber := 42.
Transcript show: ’21 + 21 is: ’, answerNumber asString.
```

6 Generating and Understanding a Trace

Now we would like to show you how we can use the Transcript to generate a trace of a program. A trace is a collection of indications that is generated by a program. To generate a trace, we introduce expressions that do not change the original execution of the program but for example display information to the transcript. Let us experiment with script 1.5 that draws a stair with steps of growing length.

Script 1.5 (*Stairs)*

```
| pica length |
pica := Bot new.
length := 10.
10 timesRepeat:
        [ pica go: length.
        pica turnLeft: 90.
        pica go: 5.
        pica turnRight: 90.
        length := length + 10 ]
```

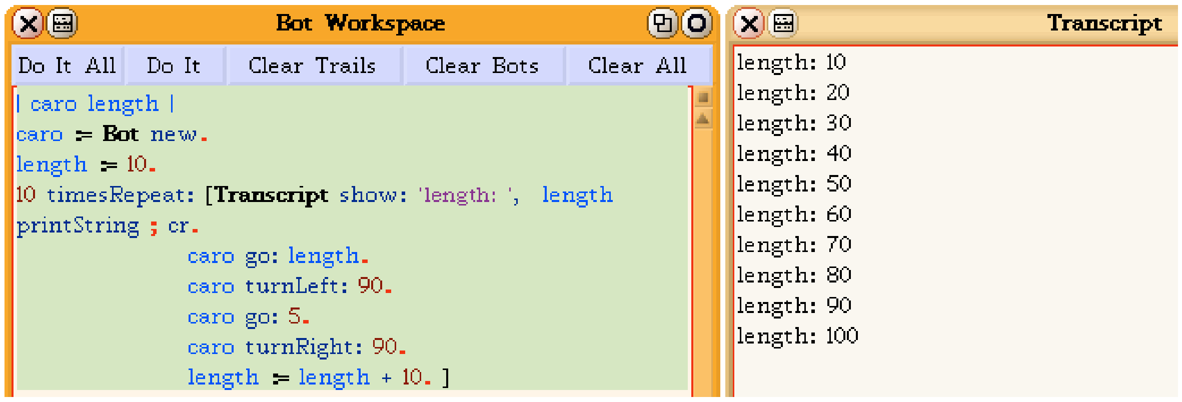
The first simple trace that we can generate is to introduce expression that display an information before and after the loop as shown in script 1.6.

Script 1.6 (*Stairs)*

```
| pica length |
pica := Bot new.
length := 10.
Transcript show: 'Before the loop' ; cr.
10 timesRepeat:
        [ pica go: length.
        pica turnLeft: 90.
        pica go: 5.
        pica turnRight: 90.
        length := length + 10 ].
Transcript show: 'After the loop' ; cr.
```

Modify the previous program and introduce the expression Transcript show: 'inside the loop' ; cr. inside the loop. You should get 10 times the 'inside the loop' information on the transcript. You can also insert the expression self halt to get a debugger.

Now we would like to use the same technique to generate a more sophisticated trace. For example, we want to see how the value of the variable length evolves while the program is executed. The following script 1.7 contains a new expression that prints the value of the variable length at the beginning of the loops each time it is executed.



```
| caro length |
caro := Bot new.
length := 10.
10 timesRepeat: [Transcript show: 'length: ',  length
printString ; cr.
              caro go: length.
              caro turnLeft: 90.
              caro go: 5.
              caro turnRight: 90.
              length := length + 10. ]
```

```
length: 10
length: 20
length: 30
length: 40
length: 50
length: 60
length: 70
length: 80
length: 90
length: 100
```

Figure 1.3: Adding a trace to a script.

Script 1.7 (*Stairs*)

```
| pica length |
pica := Bot new.
length := 10.
10 timesRepeat:
        [ Transcript show: ’>> ’,  length asString ; cr.
        pica go: length.
        pica turnLeft: 90.
        pica go: 5.
        pica turnRight: 90.
        length := length + 10 ]
```

Adding a trace after assignments is often interesting and reveals some key behavior of a porgram. For example, we suggest you to add the expression Transcript show: ’After := ’ , length asString ;cr. after the last expression of the loop as in the script 1.8. The trace shows the value of the variable length at the beginning and the end of the loop.

Script 1.8 (*Stairs)*

```
| pica length |
pica := Turtle new.
length := 10.
10 timesRepeat: [ Transcript show: 'length: ',  length asString ; cr.
        pica go: length.
        pica turnLeft: 90.
        pica go: 5.
        pica turnRight: 90.
        length := length + 10.
        Transcript show: ' length after := ' , length asString ; cr. ]
```

```
length: 10
length after := 20
length: 20
length after := 30
length: 30
length after := 40
length: 40
length after := 50
length: 60
length after := 60
length: 70
length after := 70
length: 80
length after := 90
length: 90
length after := 100
length: 100
length after := 110
```

Summary

- A string is a sequence of characters delimited by single quotes '. A string represent textual information such as words or sentences and is used to display information to the user. 'squeak is cool' is a string of 14 characters.
- A character is one letter prefixed by the dollar sign $. $a is the character a.
- A string can represent a number but this is not a number. For example, the string '69' is composed of the two characters: $6 and $9. To obtain the string representing a number, send the message asString to it.
- The Transcript is a small window used to display message for the user. The message show: aString displays the argument into the Transcript. The message cr adds a new line in the Transcript.