Control Structures - For

A for statement causes a block of expressions to be execute several times. Each time through the block of expressions, a local variable is set to a different number. For example:

```
$ s.

s := 0.

for 1 to 10 do [ $ index i.

s := s + i.

].

return s.
```

fi 55

This sums the numbers from 1 to 10. Each time through the loop (each iteration), the local variable i is set to a number from 1 to 10. Then the expression adding it to s is executed. The variable i is called the index variable.

»
The declaration of the local variable i requires the bold word index. Actually, i is an argument (a type of local variable) to the block. Arguments are discussed later. For now, just use this kind of declaration for the local variable in a for loop.

For Parameters

The for statement can take an optional argument, by, that determines how much to bump the index variable each time. Before executing the next example, open the transcript window.

To open the transcript window:
•

Tap on Transcript… in the System menu. The transcript window opens. You can move and resize this window as needed.

Execute the following example:

```
for 1900 to 2000 by 10 do [ $ index year.

$ is-mult-of-4, is-century.

is-mult-of-4 := year % 4 == 0.

is-century := year % 100 == 0.

if (is-mult-of-4 and (not is-century))

then [ transcript put year.name && "is a leap-year" ]

else [ transcript put year.name && "is not a leap-year" ].

].
```

The block of expressions gets executed once for each decade from 1900 to 2000. The transcript will show these years. This example has several important things to notice:

- The index variable can be called anything you like. In this case it was called year.
- Inside a block, there can be additional local variables.
- The % operator computes the remainder of the division between its two arguments. 12 % 5 is 2 because the remainder of 12 divided by 5 is 2.

•
The put message, used with transcript as the receiver, appends a string to the transcript window. This is useful for debugging, and seeing how your scripts are running.

Local Variables in Blocks

In the example above, two local variables were declared inside the block of the for loop. These variables are available only within the block. They are created fresh each time the block is executed, each time through the loop. After the block is done, the variables are no longer available. This is true of the index variable as well. For example:

```
for 10 to 20 do [ $ index i. ]
an empty block

"the final value of i is" && i.name.
```

This code won't execute. If you try to, the system will give you the error message "Unknown local or property" because the index variable i is not valid outside the block.

What happens if the name of a local variable is the same as the name of a variable outside the block? While this isn't recommended, it works the following way: Inside the block, the name refers to a new local variable; outside the block it refers to the variable it would if the block wasn't there. For example:

```
a := "Hello".
this sets the workspace variable
 for 1 to 10 do [ $ index i.

$ a.
declares a local variable

a := i * i.
sets the local variable

].

a := a && "kitty".
access and sets the workspace variable
```

»
A word about speed: Don't worry about the expense in time of the system creating local variables each time through the loop. Creating local variables is essentially free. In fact

variables outside the block are a little more expensive, as well as harder to read. The moral: use local variables in blocks whenever possible.