Control Structures - For

There is another form of the for statement. Instead of executing a block once for each number in a sequence, it executes the block once for each element of a group. Up to now the only examples of a group that has been discussed are strings. A string is a group of characters. Groups are actually a more generalized object that will be discussed in a later chapter. Everything here will apply to those groups as well.

In a numeric for statement you specified the parameters of the numeric sequence with the receiver, to and (optionally) the by arguments. In this for statement you specify the group of elements to iterate over, in this case a string:

```
$ text, num-vowels.

text := "John Jacob Jinglehiemer-Schmidt".

num-vowels := 0.

for text do [ $ element e.


if ("aeiou" has e)



then [ num-vowels := num-vowels + 1 ].

].

return text.name && "has"    && num-vowels.name && "vowels".
```

Another difference from previous for statements is that instead of the index variable, there is a different variable in the iteration. In this example, the local e will be set to each letter (or element) of the string in turn. The index variable is also available if you want it:

```
$ text, last-vowel
```

```
text := "John Jacob Jinglehiemer-Schmidt".

last-vowel := ???.

for text do [ $ element e, index i.

if ("aeiou" has e)

then [ last-vowel := i ].

].

if (last-vowel is-not ???)

then [ return "The last vowel is in position" && last-vowel.name ]

else [ return "There were no vowels" ].

»
```

By now, you know enough about scripts to recognize that declaration expression in the for statement's block looks exactly like argument declarations for a script. In fact they are argument declarations. The for statement passes arguments to the block, and element and index are block arguments.

Reverse For

This form of for statement has an optional argument reverse. If you specify reverse, then the elements (and the index) are iterated from the end to the start:

```
$ text, rev-text.

text := "college".
```

```
rev-text := "".

for text reverse do [ $ element e.

rev-text := rev-text & e.

].

return rev-text.name && "is" && text.name && "spelt backwards".
```

»

Notice that the reverse argument doesn't take any value, you just add it to the statement. This is an example of a flag argument, which is discussed in more detail later.

»

Numeric for statements don't use reverse. Instead you just use a negative by argument:

```
for 10 to 1 by -1 do [ $ index t.

transcript put "T minus" && t.name & ", and counting…".

].

transcript put "Liftoff!".
```