Conditional statements

Most conditional statements in Visual Basic for Applications (VBA) have AppleScript equivalents.

if / then / end if, and if / else / then / end if

The following two statements are used for the same condition:

AppleScript

```
if...then
end if
```

VBA

```
If...then
```

In AppleScript, if the statement is just one line, you can type it on a single line rather than in a block, and omit the `end if`. Compare the following examples:

AppleScript single line conditional statement formats

```
if x = 3 then
        set y to 4
end if
```

```
if x = 3 then set y to 4
```

Similarly, the following AppleScript statement is the same as that used for the same condition in VBA:

AppleScript

```
if...
else if...
else if...
else if...
end if
```

There is no `Select Case` statement, so you use this `if / else if` construction instead.

tell / end tell

The equivalent of the VBA `with / end with` statement in AppleScript is the `tell / end tell` statement targeted at an application object, as shown in the following example:

```
tell application "Microsoft Word"
        set textObject to the text object of active document
        tell (get find object of textObject)
                set forward to true
                set wrap to find continue
                set format to false
                set match all word forms to false
                set match case to false
                set match sounds like to false
                set match whole word to false
                set match wildcards to false
                clear formatting
                clear all fuzzy options
                clear formatting (its replacement)
        end tell
end tell
```

The code below illustrates the standard method for setting options in VBA. You use a `with / end with` block. The following example uses the **Find** object in Microsoft Word:

```
With ActiveDocument.Range.Find
End With
```

However, in AppleScript, you can go a step further to condense the script by using a list and a `tell` block to set multiple properties. The following example uses the **find** object:

```
tell (get find object of textObject)
        set {forward, wrap, format, match all word forms,
                match case, match sounds like, match whole word,
                match wildcards} to {true, find continue, false,
                false, false, false, false}
        clear formatting
```

```
        clear all fuzzy options
        clear formatting (its replacement)
end tell
```

You cannot use `of` to set multiple properties; however, you can access them in a list using **get** and `of`, as shown in the following example:

```
get {forward, wrap, format, match all word forms,
        match case, match sounds like, match whole word,
        match wildcards}
        of find object of text object of active document
```

repeat / end repeat

The equivalent of VBA's `Do / Loop` and `For / Next` loops in AppleScript is the `repeat / end repeat` block. There are five types.
Most commonly, in place of VBA's `For Each / Next` loops, you will use the following AppleScript:

```
repeat with eachItem in aList
-- code here
end repeat
```

This construction works when `eachItem` is an application object reference. When `eachItem` is a reference that has not yet been evaluated, this construction doesn't work. The = equality operator behaves unexpectedly, as shown in the following example:

```
repeat with eachItem in {1,2, 3, 4, 5}
        if eachItem = 2 then
                beep
                display dialog (eachItem as string)
                exit repeat
        end if
end repeat
display dialog (eachItem as string)
```

The result of this code is that no beep is heard, and the dialog displays 5. That's because `eachItem` never equals 2. It equals item 2 of `{1,2,3,4,5}`. You can solve this problem by replacing the second line with the following code:

```
if contents of eachItem = 2 then
```

You can also solve it using strategies that force an evaluation of `eachItem`, such as coercion. However, it is simpler to use the following construction:

```
repeat with i from 1 to (count {1, 2, 3, 4, 5})
        set eachItem to item i of {1, 2, 3, 4, 5}
        if eachItem = 2 then
```

This construction creates a `For / Next` loop in all cases.
There is an important difference between VBA and AppleScript when using these loops. The `For / Next` loop in VBA looks like the following example:

```
For i = 1 to 5...
Next i
```

When you exit the loop after five loops, the value of `i` is 6. However, in AppleScript, the value of `i` is 5 when you exit the following loop:

```
repeat with i from 1 to 5
```

This result will surprise VBA coders who rely on reading the value of `n` to see whether a loop was exited normally, or by means of an `Exit For` statement. In AppleScript, `exit repeat` is the equivalent of `Exit For`, and gets you out of the block.

try / on error / end try

There is a construction in AppleScript that is similar to the `On Error` construction in VBA. You must frame the section of code in which you want to trap the error with a `try / end try` block, as shown in the following example:

```
try
--code here
end try
```

This code is like using an `On Error Go To` line after `end try` for any runtime error that occurs within the block.
The following code provides an alternative construction in the event of an error for which you use `On Error Resume Next` in VBA, but it can be located anywhere in your script:

```
try
--code here
on error
--alternate code here
end try
```

The following more elaborate construction lets you branch depending on what the specific error number or message is:

```
try
--code here
on error errMsg number errNum
        if errNum = -1708 then
                --alternate code here quoting or parsing
                --the message and/or checking for the
                --error number
        else
                -- do something else
        end if
end try
```

return / error number -128

To quit the top level of a script, you can use `return` at any time. In a subroutine, that will get you back to the main script after the line that called the subroutine. To quit the whole script, even from a subroutine, use the following code:

```
error number -128
```

This construction is the same as using the Cancel button. Be sure you're not in a `try` block at the time. If you are in a `try` block, you have to test for the error number, and if it is `-128`, you then invoke `error number -128` a second time in the `on error` section.