Subroutines and functions

Implicit run handler in AppleScript

In a Visual Basic for Applications (VBA) macro, every line of code is in a subroutine or function. Even when there's only one simple procedure in a macro, it has to be in a named subroutine, such as `Sub Whatever()`. In AppleScript, the equivalent to "one simple procedure" is the top-level of a script, not a subroutine.
The top level of a script is actually an implicit `run` handler, but you can make it explicit by encasing it all in an `on run / end run` block. However, there's seldom a reason for doing so.
There is one rare situation in which to use the block: when you load and call one script from another external script, and you need to pass it arguments. Typically, you accomplish this task by calling a subroutine. But there are some rare circumstances in which you can run a script, but you can't call a subroutine, such as calling from an application, shell, or Internet context outside of AppleScript.
It is also customary, although not necessary, to include an `on run` block when you create a "droplet" and you want a separate procedure to engage when it is double-clicked. A droplet is a script application that is activated by dropping files or folders onto it.
For the most part, however, when you convert a VBA macro that has one `Sub()`, you only need the top-level script.

Calling subroutines and functions

What about more complex VBA macros that call other subroutines or functions? The construction is virtually identical in AppleScript and VBA, except that you don't use the word `Sub` or `Function` in AppleScript. Instead, the parentheses after the handler's name tell AppleScript that you're calling a handler.

Note There is another way to call and name handlers without parentheses, called "labeled parameters," but they are seldom used.

In AppleScript, there's no difference in syntax between a VBA `Sub`, which is a routine that doesn't return a result, and a VBA `Function`, which is a routine that returns a result. Compare the following examples:

VBA

```
SubroutineA
```

```
Call SubroutineA
```

AppleScript

```
SubroutineA()
```

Calling handlers within application tell blocks

However, in AppleScript, when you call a handler from within an application `tell` block, which you do regularly, you must precede the call with the `my` keyword; otherwise, you receive an error. The `my` keyword tells AppleScript that `SubroutineA` belongs to the script itself, and not to the application, such as Microsoft Word, as shown in the following example:

```
tell application "Microsoft Word"
        my SubroutineA()
end tell
```

Another option is to use the `of me` synonym after the handler name, as in: `SubroutineA() of me`.
You can use `my` even when you're not in a `tell` block. If you get in the habit of using it every time, you can't go wrong.
The construction is similar when you pass parameters to a subroutine. Compare the following examples:

VBA

```
SubroutineA param1, param2
```

```
Call SubroutineA(param1, param2)
```

```
theResult = FunctionB(param1, param2)
```

```
Set theResult = Call FunctionB(param1, param2)
```

AppleScript

```
my SubroutineA(param1, param2)
```

```
set theResult to my SubroutineA(param1, param2)
```

In summary:

- `SubroutineA` is followed by parentheses
- `SubroutineA` is not preceded by the word `Sub` or `Function`
- `SubroutineA` is followed by the obligatory `on`, or its synonym, `to`

The following example illustrates what your code should look like:

```
on SubroutineA(param1, param2)
        display dialog "Here's " & param1 &
                " and " & param2
end SubroutineA
```

Sending commands to applications in nested tell blocks

If you are sending commands to Word, Microsoft Excel, or any other application inside a subroutine, you must include a new `tell` block to that application, even if the call to the subroutine was itself within a `tell` block to the very same application. The "calling" `tell` block has no force inside another subroutine (handler).
Most often, especially when you are converting VBA macros, both the calling top-level script or subroutine and the called subroutine send commands to Word, Excel, or Microsoft PowerPoint, and both subroutines need their own `tell` block to that application.

Subroutines and functions are the same in AppleScript

Note that in AppleScript, function handlers that return results are no different than ones that don't. For example, in the following table, the handler in the first code sample could be called by the second:

Subroutines and handlers look the same

```
on SubroutineA(param1, param2)
        set someText to "Here's"
                & param1 & " and " & param2
        return someText
end SubroutineA
```

```
set theResult to my SubroutineA(param1, param2)
```

Modules

People working on large-scale VBA projects, particularly projects involving several collaborators, may be used to controlling multiple VBA modules from a central module. When you use a central module, global variables are passed down to subsidiary modules, written by multiple collaborators.
A similar modular structure is available in AppleScript using script objects. Script objects are considered an advanced AppleScript feature because they are not needed much in basic scripting; however, they are fundamental to AppleScript.
AppleScript files (.scpt, .app) that you use to save your scripts are one form of script object. They can be loaded by other scripts using the **load script** command that is located in the Standard Additions library. When you load a script, all of its properties, variables, and handlers are available to the calling script, which can set its script properties and variables to these for convenience.
Therefore, one script can act as a master script, loading the collaborators' script files as script objects, which are used in the same way as that VBA uses modules.