Other language differences

There are four primary Visual Basic for Applications (VBA) for Microsoft Office features that don't exist in AppleScript. There are also a couple of powerful AppleScript features that don't exist in VBA.

Scripts aren't located in application toolbars

You cannot run scripts from toolbars in Office applications. However, you can run scripts from the Script menu. There are third-party utilities that provide palettes and toolbars with buttons that run scripts. You can also make your own toolbars.

Auto macros don't run automatically on open and close

You can't run auto macros automatically when opening or closing a document or an application. In order for auto macros to work, scripts have to be stored in the documents themselves, as macros are now, and there have to be event procedures that watch the application and intercept buttons and menu items.
AppleScript is capable of implementing events. For example, there are Folder Actions in the Finder that know when a file is added to a folder. However, most applications do not provide events.
Events give a scripter the ability to interfere with the application. Storing scripts in documents, which is essential for auto macros to function, allows macro viruses to infest computers.
While you can't convert your auto macros to auto scripts, you can make them into regular scripts that you run manually after launching an application or opening a document. You can also have a Microsoft Entourage schedule run scripts for Microsoft Word, Excel, or PowerPoint using cron, osascript, or scheduled via iCal.

No UserForms

UserForms is not available in AppleScript, but there is a solution.
Macintosh AppleScript Studio is a robust tool for creating Cocoa applications using AppleScript. Visual objects are mostly made using drag and drop objects and controls, which you control using a specialized AppleScript dictionary that contains many Cocoa classes and methods that have been translated into AppleScript. AppleScript Studio is part of the Mac OS X Xcode Developer Tools package, which comes free on the Mac OS X installation CD.
If you use AppleScript Studio, you have access to AppleScript that is already written for Office and other applications. For more information about AppleScript Studio, see the AppleScript home page (http://developer.apple.com/applescript/) and AppleScript Fundamentals (http://developer.apple.com/referencelibrary/API_Fundamentals/AppleScript-fund-date.html) on the Apple Developer Connection Web site.
You may also find third-party "scripting additions" that can help with replacing UserForms.
Scripting additions extend the AppleScript language, much like plug-ins. You can use scripting additions when you install them in a folder that you create and name ScriptingAdditions at /Library or /Users/*username*/Library. Apple has already installed its own collection called Standard Additions, located at /System/Library/ScriptingAdditions. Check out the Standard Additions dictionary.
If you use third-party scripting additions, you must do the following:

1. Save your script as a script bundle or application bundle.
2. Create a Scripting Additions folder inside the bundle's Resources folder, located at /Contents/Resources.
3. Distribute the bundle.

You must always check for licensing requirements, of course.
An advantage of using AppleScript Studio to create scripting additions over other tools is that the frameworks required to run the distributed applications are built into every Macintosh or are included in the builds. Users don't have to install anything else.

No AppleScript recorder

Another VBA feature that is not available in AppleScript is an AppleScript recorder. However, script recorders have their limitations because they simply mirror user interface actions using 'selection,' and they don't provide conditionals or other robust functionality.

AppleScript whose filters

Meanwhile, AppleScript has some features that VBA does not, including the ability to connect to almost everything else on your Macintosh. Most importantly, there are two features, which are implemented by most applications, that can really speed up and simplify your scripts.
The first is the `whose` (or `where`) filter. Macintosh Script Editor used to refer to it as specifying an element "by satisfying a condition." The following example uses the `whose` filter:

```
tell application "Microsoft Word"
        every paragraph of active document
                whose content of text object
                contains "AppleScript"
end tell
```

This code returns a list of the 66 paragraphs in this article that contain the string "AppleScript," in two seconds flat. You don't have to create a laborious `repeat` loop that works through every paragraph. You also have the option to ask for `first paragraph` and `last paragraph`, but that version of this code has to be in a `try / on error` block, or you will receive an error if there are no paragraphs.
The `whose` filter is a powerful time- and effort-saving tool that works on virtually every element of every class in all the applications. You can replace your `For`, `For Each`, and `Do While` repeat loops in VBA with `whose` filters when you convert your macros to AppleScript. Another good reason to switch is that `repeat` loops seem to run more slowly in AppleScript than in VBA.
There is one pitfall that you should be aware of when using the `whose` filter. If the name of the property that you're filtering on also happens to be the name of a class, such as a **category** in Entourage or **replacement** in Word, use the synonym `where its` instead of `whose`. In AppleScript, when the context in unclear, class names have priority. When the element and property have the same name, the word `its` focuses AppleScript on the correct one. Otherwise, you can get an incorrect result of `{}` (no results).

AppleScript property lists

Another AppleScript feature that helps to eliminate tedious `repeat` loops is to ask for the (singular) property of every element, as shown in the following example:

```
tell application "Microsoft Word"
        name of every document
end tell
-->{"Document3","MacPowerPointVBA_PunchList.doc",
"Converting VBA To AppleScript in Microsoft Office.doc",
"Document2"}

        name local of every Word style of active document
        --> {"1 / 1.1 / 1.1.1", "1 / a / i", "Article / Section",
"Balloon Text", "Block Text", "Body Text", "Body Text 2",
...etc
```

This code returns a `list` of the requested property for every element. There is no need to use a `repeat` loop through each element. For example, with Word styles, there are upwards of 165 elements. In other contexts, there are hundreds or thousands of elements.
Very occasionally, you will find a bug with this method. For example, an error could occur when you try to get `content of every word of text object of paragraph 1 of active document`, wherein the bug is actually with `every word`, and not with the `content of property` retrieval. That said, bugs are very rare. If you can get `every element of some object` as a `list`, you can also get any of its properties in a list, too.

Combining whose filters and property lists

Note that you cannot apply a `whose` filter to the resulting list, or to any other AppleScript list. You can only use it with application elements. That said, most applications, including Entourage, allow you to combine these two features to get a property list on a `whose` filter, as shown in the following example:

```
tell application "Microsoft Entourage"
        name of every contact whose default email address
                contains "microsoft.com"
end tell
--> {"Jay Jamison", "Holly Holt", "Parry Bedi"}
```

However, you can't perform the equivalent task for Word, Excel, or PowerPoint, as shown in the following example:

```
tell application "Microsoft Word"
        name of every document whose content of text object
                contains "AppleScript"
end tell
--> missing value
```