Excel AppleScript

Microsoft® Excel®:mac 2008

このドキュメントに記載されている情報 (URL などのインターネット Web サイトに関する情報を含む) は、情報提供の目的で発行されているものであり、将来予告なしに変更することがあります。

別途マイクロソフトのライセンス契約上に明示の規定のない限り、このドキュメントはこれらの特許、商標、著作権、またはその他の無体財産権に関する権利をお客様に許諾するものではありません。

Microsoft、Excel、Entourage、PowerPoint は、米国 Microsoft Corporation およびその関連会社の登録商標または商標です。

Apple、Apple ロゴ、Mac、Mac OS は、米国 Apple Inc. の米国およびその他の国における登録商標または商標です。

© 2008 Microsoft Corporation. All rights reserved.

Excel AppleScript

Excel AppleScript の基本

Microsoft Excel AppleScript トピックでは、Visual Basic for Applications (VBA) で一般的によく使用されるサブルーチン (マクロ) と、これに相当する AppleScript サブルーチンについて説明します。

Macintosh スクリプト エディタ でスクリプトをテストするときは、各スクリプトの `tell application "Microsoft Excel"` ブロックの先頭に単独行として **activate** コマンドを指定することができます。これにより、Excel が前面に表示されるため、動作を確認しやすくなります。Excel が前面に表示されているときに、保存したスクリプトを [スクリプト] メニューから実行する場合は、**activate** を指定する必要はありません。他の場所からスクリプトを実行していて、Excel を前面に表示する必要がない場合も同様です。各トピックで紹介するサンプル スクリプトには **activate** コマンドは含まれていませんが、必要に応じて使用することもできます。

Excel でスクリプトの作成を開始する前に、Excel 2004 AppleScript リファレンス (http://download.microsoft.com/download/E/0/6/E06153BE-8B14-4A13-9F35-109218A9CCCB/Excel2004AppleScriptRef_J.pdf) の「Excel Dictionary の使用」(特に「セルとセル範囲の参照方法」) をお読みになることをお勧めします。このドキュメントは、AppleScript よりも Excel オブジェクト モデルに慣れていないユーザーを対象にしていますが、すべての Excel スクリプトの作成者に役立つ情報が記載されています。

Excel は最も複雑で精巧なアプリケーションの 1 つであり、さまざまな機能が盛り込まれています。各トピックでは、広範なテクニックを説明するために、比較的短い VBA コマンドを AppleScript に変換する例を紹介しています。実際のマクロは、それらのコマンドを組み合わせて記述されるため、個々のテクニックを習得すれば、さまざまな状況で応用することができ、AppleScript Dictionary をさらに活用できるようになります。

VBA で記述された多種多様な Excel 機能のほとんどは、直接 AppleScript に変換できますが、メソッドやプロパティの中には、変換できない、または形式の異なるものもあります。トピックでは、そのような場合の解決方法や回避策についても説明しています。

具体例を挙げると、Excel の AppleScript 実装は、ほとんどの VBA モデルが AppleScript モデルで正確に再現できるレベルまで徹底して行われていますが、列と行に関しては特有の問題があります。詳細については、「Excel 2004 の既知の問題」を参照してください。

また、ワークシート関数が VBA から AppleScript に正しく変換されないという問題もあります。Excel によるワークシート関数の処理と、異なる浮動小数点数精度を使用する AppleScript による処理には、明らかに差異が生じます。しかし、シート上のセルに (AppleScript を使用して) 任意の関数を入力して Excel で計算し、結果 (セルの値) をスクリプトに返してセルをクリアするという方法で対処することができます。

ブックを開く

ブックを作成する

空白の新しいドキュメントを作成する場合、VBA では次のコードを使用します。

```
Application.Workbooks.Add
```

AppleScript では次のようになります。

```
make new workbook
```

AppleScript では通常、作成時に **with properties** パラメータに **workbook** のプロパティの大半を指定できます (スクリプト作成者は、必要な要素だけを後で追加することになります)。このようなことは、Microsoft Word 文書ではできませんが、Microsoft Excel では可能です。

Dictionary に読み取り/書き込みと記載されているすべてのプロパティには、共有ブックとして最初に設定が必要な場合を除き、**make new** コマンドで、`with properties {...}` を設定できます。 Visual Basic for Applications (VBA) では、新しいブックを作成し、変数でブックへの参照を設定してからこの設定を行う必要があります。

AppleScript でも同じことができますが、さらに、次に示すプロパティを最初に設定できます。ここでは、プロパティはすべて既定値以外の値に設定され、**password** の設定を含め、すべて正常に動作します。

```
tell application "Microsoft Excel"
        set newWkbk to make new workbook with properties  ¬
                {acceptlabels in formulas:false, ¬
                        conflict resolution:¬
                        local session changes, date 1904:¬
                        false, display drawing objects:¬
                        placeholders, personal view list settings:¬
                        false, personal view print settings:¬
                        false, precision as displayed:true,
                        save link values:false, ¬
                        update remote references:¬
                        false, template remove external data:¬
                        true, keep change history:¬
                        true, remove personal information:¬
                        true, password:"yyyyy", write password:¬
                        "zzzzz", read only recommended:¬
                        true, workbook comments:"Hi, there"}
end tell
```

これらが既定の設定でないことを確認するには、まず通常の空白の新規ブックで、次のコードを実行します。

```
properties of active workbook
```

この実行結果をテキスト エディタにコピーしておきます。次に、既定値以外のプロパティ値を設定する先のスクリプトを使用して別のブックを作成した後、このコードを再度実行します。この実行結果を比べてください。ブックを保存して再び開くと、パスワードの入力が必要になります。その他のプロパティは、[計算方法] ダイアログ ボックスに記録されているプロパティも含めてすべて設定済みです。Excel で設定を確認するには、使用するバージョンによって多少異なりますが、次の操作を行います。

- [Excel] メニューの [環境設定] をクリックし、 [数式とリスト] の [計算方法] をクリックします。

make new コマンドで最初にプロパティを設定すると時間の節約になります。今後、AppleScripts を記述する際は、この方法を検討してみることをお勧めします。VBA マクロを変換する場合は、標準の VBA マクロを行単位で変換していく方が簡単です。

```
Dim newWkbk As Workbook

        Set newWkbk = Workbooks.Add
        With newWkbk
                .AcceptLabelsInFormulas = False
                .PrecisionAsDisplayed = True
                ' etc.
End With
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        set newWkbk to make new workbook
        tell newWkbk
                set accept labels in formulas to false
                set precision as displayed to true
                -- etc
        end tell
end tell
```

`Dim` ステートメントは、AppleScript にはないので省略します。

テンプレートからブックを作成する

Workbooks.Add メソッドのオプションの引数の 1 つである **Template** は、特定のテンプレートに基づいて新しいブックを開くときに指定するものですが、AppleScript の **make new** コマンドには、これに対応する **template** パラメータがありません。また、ブックで最初に設定する **template** プロパティもありません。

ただし、この問題を解決する非常に簡単な方法があります。Microsoft Word 2004 for Mac 以前のバージョンと異なり、Microsoft Excel テンプレートを開くと、テンプレートから作成された新しいドキュメント (**workbook**) が開きます。Excel、Finder、Visual Basic for Applications (VBA)、AppleScript のいずれから開いても同じです。

したがって、`make new document` を実行する代わりに、テンプレートを開きます (**open**)。VBA のコードを次に示します。

```
Application.Workbooks.Add
Template:="Mac HD:Folder:Template.xlt"
```

これを次のように変換します。

```
tell application "Microsoft Excel"
        open "Mac HD:Folder:Template.xlt"
end tell
```

これで完了です。

Standard Suite の基本コマンド **open** は結果を返さないので、新しいブックに変数 (参照) を設定する必要がある場合は、代わりに Excel Suite の Excel **open workbook** コマンドを使用してください。

```
tell application "Microsoft Excel"
        set newWkbk to open workbook workbook file name ¬
                "Mac HD:Folder:Template.xlt"
end tell
```

実行する場合は、こちらの方法をお勧めします。

既存のブックを開く

既存の **workbook** を開くには 2 つの方法があります。すべてのアプリケーションには、最も基本的なコマンドの 1 つ、Standard Suite の **open** コマンドが実装されています。この **open** コマンドは、スクリプトに対応していないアプリケーションを含むすべての Macintosh アプリケーションに必要です。**open** の他、**run**、**print**、および **quit** も必須コマンドとして実装されています。アプリケーションによっては、**open** に独自のパラメータが追加されることもあり、Microsoft Word がその一例です。

一方、スクリプトに対応していないアプリケーションや Finder でも、**open** を使用することで指定方法を意識することなくあらゆるアプリケーションのドキュメントを開くことができます。この柔軟性を最大限に保つため、コマンドでは結果が返されないようになっています。このことは、スクリプト対応アプリケーションが自身のドキュメントを開くときに問題になります。

この問題の良い解決策として、Microsoft Excel では、Standard Suite の **open** がファイルの一覧全体を一度に開くためのコマンドとしてそのまま採用されています。このコマンドに特別なパラメータはありません。そして、これとは別に Excel Suite の **open workbook** コマンドが用意されています。これは Visual Basic for Applications (VBA) の **Open** に相当し、パラメータ (引数) もすべて同じです。このコマンドは結果を返します。したがって、**open workbook** コマンドに変数 (参照) を設定することができます。これが、最良の方法です。

```
Workbooks.Open Filename:="Mac HD:Folder:File.xls"
```

これは次のように変換されます。

```
tell application "Microsoft Excel"
        open alias "Mac HD:Folder:File.xls"
        -- or:
        open "Mac HD:Folder:File.xls"
        -- or:
        open workbook workbook file name "Mac HD:Folder:File.xls"
end tell
```

一方、次のコードの場合は、

```
Set oWkbk = Workbooks.Open(Filename:="Mac HD:Folder:File.xls")
```

次のように変換されます。

```
tell application "Microsoft Excel"
        set theWkbk to open workbook ¬
                workbook file name "Mac HD:Folder:File.xls"
end tell
```

open workbook では多くのオプションを指定できます。たとえばテキスト ファイルを開くには、次のように引数 **Format** をとるマクロを使用できます。この引数は区切り文字 (1=Tab, 2=Comma, 3=Space など) を指定するものです。

```
Workbooks.Open FileName:="Mac HD:Folder:File.txt", Format:=1
```

これは次のように変換されます。

```
open workbook workbook file name ¬
        "Mac HD:Folder:File.txt" format tab delimiter
```

format パラメータには 1、2、3 のように番号を指定することもでき、スクリプトは問題なく動作します。VBA マクロから変換する場合は、わざわざ Microsoft Visual Basic Editor を開いてオブジェクト ブラウザ内を探し、どの番号がどのパラメータに対応するかどうかを確認する必要はなく、スクリプトで同じ番号を使用できます。このような番号で指定できる列挙値は多くあります。ただし、スクリプトを最初から作成する場合は、固有名を使用する方が便利です。

テキスト ファイルの場合は、さらに強力な **open text file** というコマンドがあります。VBA ではこれに対応する **OpenText** で VBA マクロを実行することができますが、AppleScript でも同じようにします。次に例を示します。

```
Workbooks.OpenText FileName:="Mac HD:Folder:File.txt", _
        DataType:=xlDelimited, Tab:=True
```

これは次のように変換されます。

```
tell application "Microsoft Excel"
        open text file filename ¬
                "Mac HD:Folder:File.txt" data type delimited with tab
end tell
```

AppleScript では、パラメータ名に続けてブール型のコマンド値である true と false を使用した場合 (**tab** *true*、**tab** *false*)、*true* は *with* にコンパイルされ、*false* は *without* にコンパイルされます。また、*without* **tab**、*with* **tab** のように、値はパラメータの前に付くこともあります。このような状況が発生しても、あわてる必要はありません。どちらでも指定することができます。

次のように、同じ値のブール値は、コンパイルするとすべてまとめられます。

```
tab true consecutive delimiter true
```

これは次のようにコンパイルされます。

```
with tab and consecutive delimiter
```

VBA の **OpenText** コマンドには、オプションの引数 **FieldInfo** で、もう少し複雑な情報が指定されていることもあります。

```
Workbooks.OpenText FileName:="Mac HD:Folder:File.txt", _
        DataType:=xlDelimited, Tab:=True, _
        FieldInfo:=Array(Array(3, 9), Array(1, 2))
```

これは特定の列のデータ型の指定に関するコードですが、この意味を正確に把握していなくても、**FieldInfo** の値を次のように書き換えることができます。

```
open text file filename ¬
        "Mac HD:Folder:File.txt" data type delimited ¬
        field info {{3, 9}, {1, 2 }} with tab
```

ここでは VBA の配列が AppleScript のリストになっています。単に変換するだけではなく理解する必要がある場合や、このメソッドに問題がある場合は、VBA ヘルプまたは Excel 2004 AppleScript リファレンス (http://download.microsoft.com/download/3/A/7/3A70FB4B-0C6A-43E3-AAB7-AC9166B25632/Excel2004AppleScriptRef.pdf、英語) で、**field info** についての説明をお読みください。次に、Dictionary に従って次のように指定します。

```
open text file filename ¬
        "Mac HD:Folder:File.txt" data type delimited ¬
        field info {{3, skip column}, {1, text format}} with tab
```

ワークシートを使用する

ワークシートまたはグラフを追加する

ワークシートを追加する

新しい **worksheet** を **active workbook** に追加して、作業中のシートの左側に配置するには、次のコードを使用します。

```
ActiveWorkbook.Worksheets.Add
```

AppleScript では次のようになります。

```
make new worksheet at active workbook
```

メモ AppleScript の `tell application "Microsoft Excel"` ブロックは省略されていますが、常に適用されます。すべてのステートメントは、Microsoft Excel `tell` ブロックにある場合にのみコンパイルが実行されます。

in の方が使いやすい場合は、**at** の代わりに使用できます。

```
make new worksheet in active workbook
```

どちらを使用しても同じように動作します (AppleScript では同じ動作を表現するさまざまな類義語や同義語があります)。

ここでも、最初に読み取り/書き込みプロパティを設定できます。

```
set newSheet to make new worksheet at active workbook with properties ¬
        {name:"Test 1", display page breaks:false}
```

Visual Basic for Applications (VBA) では、**Before** または **After** を使用すると、新しいシートを正確に配置できます。VBA で 1 つのシートをブックの先頭に追加するには、次のコードを使用します。

```
ActiveWorkbook.Worksheets.Add Before:=1, Count:=1
```

AppleScript では、次のようになります。

```
make new worksheet at beginning of active workbook
```

2 つのシートを最後のシートの右側に追加するには、次のコードを使用します。

```
With ActiveWorkbook.Worksheets
        .Add After:=.Item(.Count), Count:=2
End With
```

AppleScript の場合、複数の要素を同時に作成する方法がないため、少し複雑になります。**make new** で作成される要素は 1 つだけです。ただし、要素を挿入する方法は多数あります。

AppleScript で挿入場所を示す方法として適切な例を次に示します。

- `at beginning of`
- `at end of`
- `at before [some element]`
- `at after [some element]`

ただし、`at beginning of` と `at end of` は問題なく動作しますが、**before** と **after** は、動作が不安定です。

要素 (ここでは既存の **sheet**) を参照せずに、単純に `make new worksheet at before active workbook` と記述すると、プロパティ **before** は予期したとおりに動作するように見えますが、実際は、**before** は無視され、シートは既定の場所 (先頭) に作成されます。たとえば、`at before sheet 3` で試してみると、1 回目は動作しますが、新しい `sheet 3` がアクティブ シートでない場合はクラッシュします。

次に示す VBA の例の場合、シート数をカウントする必要はなく、新しいシートを末尾に 2 回作成します。

```
repeat 2 times
        make new worksheet at end of active workbook
end repeat
```

たとえば、既に 4 つのシートがあり、どれがアクティブ シートか不明の状態で (左側の) 最初のシートの後に 2 つのシートを挿入する場合は、**after** を使用できます。ただし、**at** は省略しません。しかし、次のコードには問題があります。

```
tell application "Microsoft Excel"
        tell active workbook
                repeat 2 times
                        make new worksheet at after sheet 1
                end repeat
        end tell
end tell
```

Excel は 2 回目の `repeat` でクラッシュします (前の説明のとおり、1 回目の `repeat` が終了すると、`sheet 1` はアクティブではなくなります)。このような場合、Standard Suite の **move** コマンドを使用して、シート間に新しいシートを作成します。

VBA の **Copy** メソッドを AppleScript に変換する際、**before** パラメータと **after** パラメータを取得するためには、Excel の **copy worksheet** コマンドが必要でしたが、Standard Suite の **move** コマンドには既に **to** パラメータがあるため、**before** と **after** を挿入できます。このように、**move** を活用できる設計になっています。

`sheet 1` の後に 2 つの新しいシートを挿入する方法を次に示します。

```
tell application "Microsoft Excel"
        tell active workbook
                repeat 2 times
                        set ws to make new worksheet at end
                        move ws to after sheet 1
                end repeat
        end tell
end tell
```

グラフを追加する

VBA で **Chart** を追加するには、**Sheets** コレクションで引数 **Type** と **Add** メソッドを使用します (**Charts** は **Worksheets** ではなく、**Sheet** の一種です)。

```
ActiveWorkbook.Sheets.Add Type:=xlChart
```

ただし、**sheet** クラスの **worksheet type** プロパティには、種類の 1 つとして *sheet type chart* が含まれますが、このプロパティは読み取り専用で、最初に設定することもできません (いくつかのコマンドにも同じことが言えます)。新しい **sheet** を作成するときに種類を *sheet type chart* に設定しようとすると、このプロパティは無視されます。その結果、通常の新しい **worksheet** が作成されます。これは想定内のことで、バグではありません。単に、AppleScript の `make new with properties` は、VBA の **Add** メソッドとは異なるということです。

AppleScript では、**chart sheet** は VBA の **Chart** に対応するクラスで、**sheet** のサブクラスです。VBA と異なり、グラフを直接作成できます。

```
tell application "Microsoft Excel"
        make new chart sheet at active workbook
end tell
```

これは完全に空白のシートで、グラフの作成準備ができている状態です (**chart** は **chart sheet** の読み取り/書き込みプロパティで、このクラス用のほとんどのプロパティと一緒に設定できます)。**Charts** は複合オブジェクトです。グラフの詳細については、「グラフを作成する」を参照してください。

1 つのワークシートに対して複数のウインドウを開く

Visual Basic for Applications (VBA) では、複数のウインドウを使用して、同じワークシートの異なる部分を表示できます。

```
ActiveWindow.NewWindow
Windows.Arrange ArrangeStyle:=xlHorizontal
```

AppleScript では、新しい **window** を作成することはできますが、**arrange** コマンドはありません。そのため、次のコードを使用する必要があります。

```
tell application "Microsoft Excel"
        tell active workbook
            set {l, t, h, w} to window 1's {left position, top, height, width}
            new window on workbook
            tell window 1 to set {left, top, height, width} to {l, t, h / 2, w}
            tell window 2 to set {left, top, height, width} to {l, (t + h / 2), h / 2, w}
        end tell
end tell
```

Height は、画面の一番上 (0, 0) を開始位置として、ポイント単位で計算されます。そのため、画面の下半分の位置を開始位置にするには、高さの半分を加算する必要があります。

active window は、参照したときに前面にあるウインドウに対する "動的な" 参照です。変数を設定した場合でも、変数は現在のアクティブ ウインドウに "移動" します。変数をアプリケーションの **active window** に設定、またはコピーしても、同じウインドウを "保持" することはできません。新しいウインドウを作成すると、それがアクティブなウインドウになり、変数は新しいウインドウを参照します。

注意を払わないと、同じウインドウに対して両方の操作を指定することになり、その結果、画面上から半分の位置に 4 分の 1 のサイズのウインドウと、最初の位置とまったく同じ位置にフル サイズのウインドウが表示されることになります。このため、新しいウインドウを作成した後、単純にブックの `window 1` と `window 2` を参照する方が簡単です。この場合、どちらのウインドウがアクティブであるかは関係ありません。

VBA では、**View** プロパティ *(xlNormalView*、*xlPageBreakPreview*、または *xlPageLayoutView)* を使用してウインドウの表示を設定できます。

```
ActiveWindow.View = xlNormalView
```

AppleScript では次のようになります。

```
set view of active window to normal view
```

ワークシート名を変更する

Visual Basic for Applications (VBA) では、ワークシートの **name** プロパティを設定することによって、ワークシート名を変更することができます。

```
ActiveSheet.Name = "New Sheet Name"
```

AppleScript では次のようになります。

```
set name of active sheet to "New Sheet Name"
```

シート名が無効である場合や既にその名前のシートが存在する場合は、エラーになります。VBA でこの問題を回避する方法の 1 つを次の例に示します。

```
Const sNAME = "New Sheet Name"
On Error Resume Next
ActiveSheet.Name = sNAME
If ActiveSheet.Name <> sNAME
Then MsgBox "Re-name failed"
On Error GoTo 0
```

AppleScript では、次の例に示すようにして問題を回避できます。

```
tell application "Microsoft Excel"
        try
                set name of active sheet to "New Sheet Name"
        on error
                display dialog "Re-name failed." buttons {"Cancel"} with icon 0
        end try
end tell
```

この例では、コードは警告なしで失敗するのではなく、トラップ可能な真のエラーを作成します。

セルの値を使用してワークシート名を変更する

セルの値を使用してワークシート名を変更することもできます。ワークシート名をセルの値に設定するには、次の例にあるコードを使用できます。

```
On Error Resume Next
With ActiveSheet
        .Name = .Range("A1").Text
End With
On Error GoTo 0
```

AppleScript では次のようになります。

```
try
        set name of active sheet to string value of range "A1"
end try
```

ヘッダーとフッターを設定する

Visual Basic for Applications (VBA) では、ワークシートの **PageSetup** オブジェクトを使用してヘッダーとフッターを設定します。

```
ActiveSheet.PageSetup.LeftHeader = Format(Date, "dd mmmm yyyy")
```

RightHeader、**CenterHeader**、**LeftFooter**、**CenterFooter**、および **RightFooter** の各プロパティも使用できます。これらのプロパティは、AppleScript では **worksheet** の **page setup object** プロパティの **center header**、**right header**、**left footer**、および **right footer** に対応します。

メモ "object" という単語がプロパティ名の最後に付いていますが、参照先のクラス名 **page setup** にはありません。

VBA では、次のスクリプトを実行して複数のヘッダーとフッターを設定します。

```
Public Sub PathAndFileNameInFooter()
        Dim wsSht As Worksheet
        For Each wsSht In ActiveWindow.SelectedSheets
                wsSht.PageSetup.LeftFooter = ActiveWorkbook.FullName
Next wsSht
End Sub
```

AppleScript では次のようになります。

```
repeat with wkSht in (get selected sheets of active window)
        set left footer of page setup object of wkSht to ¬
                full name of active workbook
end repeat
```

`repeat with someVariable in someList` の形式は、通常の VBA ループ構文にほぼ対応するものとして好んで使用される傾向にありますが、変数 `someList` をリストに設定するのではなく繰り返しステートメント (`selected sheets of active window`) の中で直接オブジェクト参照にアクセスしている場合は、**get** を明示的に指定して評価を強制実行する必要があります。

数値インデックスを使用してワークシート名を変更する

Microsoft Excel で **worksheet** をコピーする場合、既定では、ワークシートは "Sheet(xxx)" という名前になります ("xxx" は、ワークシートが追加されるたびに 1 つずつ増加する数値です)。別の名前で再作成する場合、VBA では、次の例にあるコードを使用できます。

```
Const sBASENAME  As String = "MySheetName"
Dim ws As Worksheet
Dim sTryName As String
Dim i As Long
Worksheets.Add After:=Sheets(Sheets.Count)
On Error Resume Next
sTryName = sBASENAME
Set ws =
ActiveWorkbook.Worksheets(sTryName)
Do Until ws Is Nothing
        Set ws = Nothing
        i = i + 1
        sTryName = sBASENAME & Format(i, " (0)")
        Set ws = ActiveWorkbook.Worksheets(sTryName)
Loop
ActiveSheet.Name = sTryName
On Error GoTo 0
```

AppleScript では、いくつかの点が異なります。

```
property sBASENAME : "MySheetName"

tell application "Microsoft Excel"
        make new worksheet at end of active workbook
        try
                set sTryName to sBASENAME
                set ws to worksheet sTryName of active workbook
                ws -- forces an error if no such worksheet
                set i to 0 -- needs to be explicit
                repeat -- no need to set ws to null at end
                        set i to i + 1
                        set sTryName to sBASENAME & " (" & i & ")"
                        set ws to worksheet sTryName of active workbook
                        ws -- forces an error if no such worksheet
                end repeat
        on error
                set name of active sheet to sTryName
        end try
end tell
```

このスクリプトの先頭には `script property` の宣言があります。この宣言は、必ずしも必要ではありませんが、多くの VBA マクロに含まれる `CONST` 宣言にほぼ相当します。そこで、`sBASENAME` の定義に注意が向けられますが、単に次のように宣言する場合と比べてみてください。

```
set sBASENAME to "MySheetName"
```

先頭に このように宣言する方が、後で名前を他のものに変更する場合、CONST 宣言と同様にどこで実行するかが非常に明確です。これが必ずしも必要ではないという理由は、スクリプト プロパティの主な利点である、実行間の永続性を利用していないためです (VBA では、まったく別のメソッドで実行します)。したがって、通常の **set** を使用するよりも 1 か所で変更を行うことができるという点だけが効率的ですが、定義が明確になることは確かです。このプロパティ宣言は、Excel `tell` ブロック上および外側に配置することに注意してください。

エラー トラップは、特に `repeat` ループと組み合わせて使用する場合、AppleScript では扱いがやや異なります。つまり、VBA の `On Error Resume Next` を使用する方法と違って、`try/on error/end try` 構造を慎重に配置する必要があります。`try/end try` ブロックを `repeat` ブロックの外側と内側のどちらに置くか、または両側に必要かどうかということを決定する必要があります。

`repeat` ブロックの内側に `if/end if` ブロックでの `exit repeat` の指定が必要になることもありますが、この場合は必要ありません。

AppleScript で `try/error` ブロックを使用しないで実行する別の方法としては **exists** コマンドがあります。これは Standard Suite にあり、Excel を含むほとんどのアプリケーションで実装されています。

```
property sBASENAME : "MySheetName"

tell application "Microsoft Excel"
        tell active workbook
                set i to 1
                repeat until (worksheet (sBASENAME & " (" & i & ")") ¬
                        exists) = false
                        set i to i + 1
                end repeat
                make new worksheet at end with properties ¬
                        {name:sBASENAME & " (" & i & ")"}
        end tell
end tell
```

このスクリプトは簡潔で読みやすく、エラーを強制するための行の追加や変数が必要ないという利点もあります。ここでは、単に VBA を文字どおりに "変換" することではなく、"AppleScript で考える" ことを行っています。AppleScript 言語に慣れるに従って、そのようなことが必要になります。

Microsoft Excel、Word、PowerPoint、および Apple "i" アプリケーションなどの、より新しいスクリプト可能な一部のアプリケーションでは、存在しないものに変数を設定しても、それ自体ではエラーが発生しない場合が多くなります (Entourage のような "従来の" AppleScript 対応のアプリケーションでは、エラーが発生する割合が多くなります)。エラーは、次回変数を使用したときに発生します。このため、両方のステートメントの後で、存在しない可能性のあるワークシートに `ws` を設定し、変数の即時呼び出し (暗黙的な **get**) によってエラーを強制する必要があります。

スクリプトの作成時には、このような呼び出しが必要かどうかを確認するか、習慣として常に指定することをお勧めします。指定しないと、エラーの発生が想定されません。エラー処理をしておかない場合、スクリプトの後半で特定のエラーをトラップしていない時点になって初めて (`ws` へアクセスが必要になる時点で初めて) エラーが発生したときに、特定した以外のエラーかどうか迷う可能性があります。

また AppleScript では、VBA と同様に増分値をとる変数に既定値 0 は設定されません。変数は未定義になります。たとえば、次のように記述したとします。

```
set i to i + 1
```

最初に i to 0 を設定しないと、スクリプトはエラーになります。i to 0 は、repeat ループが開始される前に初期化する必要があります。

基本的な AppleScript には、VBA の **Format** 関数に相当するものがありません。相当する関数は、サードパーティから多くのスクリプトの形で提供されています。AppleScript では、Standard Additions に組み込みの **do shell script** コマンドを通じて、強力な UNIX ツールをすべて使用できます。このツールは、必要に応じて **Format** よりも優れた機能を発揮することができます。

この場合は、単に増分変数 i を 2 つのかっこで囲んで文字列式に連結するだけです。連結演算子 & の左側の文字列により、右側の数値は強制的に文字列型に変換されるため、明示的な強制型変換 as string は必要ありません。

存在しないシート名を検出すると repeat ループを抜けて外側の on error ブロックに入るというエラー処理を行っているため、repeat ステートメント内に while や until のような条件は必要ありません。同様に、どの if ブロック内にも exit repeat は必要ありません。毎回必ず on error ブロックに入ってスクリプトの最後にたどり着きます。try/on error ブロックを構造的に使用することで、コード作成が簡単になることもあります。

最後に、AppleScript では、VBA の Nothing に相当する null へのアプリケーション参照をとる変数を設定する必要はありません。このため、これらを準備する必要もありません。以上で完了です。この例の短いスクリプトでも、VBA と AppleScript の構文にはかなり多くの違いがあることがわかります。

シートを削除する

Visual Basic for Applications (VBA) では、**Delete** メソッドを使用してシートを削除します。AppleScript では、Standard Suite の **delete** が同様に動作しますが、このコマンドでは各種項目のリストを一度に削除できるため、この動作が実装されているアプリケーションでは repeat ループを使用する必要はありません。ただし、ブックには少なくとも 1 つの **worksheet** が必要なため、ブック内の最後のシートを削除することはできません。VBA で最初のシート以外のすべてのシートを削除するには、次のように記述します。

```
On Error GoTo Err_Handler
        Application.DisplayAlerts = False
        For i = Worksheets.Count To 2 Step -1
                Worksheets(i).Delete
        Next i
        Err_Handler:
Application.DisplayAlerts = True
```

AppleScript の repeat ループでは、常に逆方向に移動する必要があります。AppleScript でこれをそのまま記述すると、次のようになります。

```
tell application "Microsoft Excel"
        set display alerts to false
        try
                set allSheets to (every worksheet in active workbook)
                repeat with i from (count allSheets) to 2 by -1
                        delete (item i of allSheets)
                end repeat
        on error
                set display alerts to true
        end try
end tell
```

しかし、わざわざこのようにする必要はありません。AppleScript を使用しているので、次のように AppleScript の方法で指定します。

```
tell application "Microsoft Excel"
        set display alerts to false
        try
                delete items 2 thru -1 of (get every worksheet ¬
                        of active workbook)
        end try
        set display alerts to true
end tell
```

これでより簡単になりました。ただし、明示的な **get** が必要なことに注意してください。try ステートメントを使用している理由は、最初からシートが 1 つしかない場合、item 2 が存在しないので、エラーが発生するためです。シートが 1 つの時点で目的が達成されているので、処理を進め、エラー メッセージ (**display alerts**) を再度有効にします。

実際は、Microsoft Excel ではエラーなしに次の操作を実行できます。

```
tell application "Microsoft Excel"
        set display alerts to false
        try
                delete (every worksheet of active workbook)
        end try
        set display alerts to true
end tell
```

Excel に指示しなくても、最初のシートはそのままの場所に残されます。エラーは発生せず、最初のシートに対する **delete** の試行も行われません。これは便利で簡単な方法ですが、すべての環境でこの方法を使用できるとは限りません。

項目のリスト全体を削除できることは、項目のリスト全体を開くことができるのと同様に、特に運用環境で、AppleScript の強力なメリットとなります。

どの場合でも、アプリケーション プロパティ **display alerts** は *false* に設定し、シートの削除のたびに Excel の安全機能による [警告] ダイアログ ボックスが表示されないようにしてください。

ワークシートを保護するまたは非表示にする

ワークシートを保護する

ユーザーが誤ってセルを変更しないように、ワークシートを保護することができますが、だれでにでも簡単に解除できるため、セキュリティ対策とはならないことに注意してください。Visual Basic for Applications (VBA) では、次のように記述します。

```
ActiveSheet.Protect Password:="drowssap"
```

AppleScript では、保護する対象によって、**protect worksheet** と **protect workbook** コマンドを使い分け、次のように記述します。

```
tell application "Microsoft Excel"
        protect worksheet (active sheet) password "drowssap"
end tell
```

また、保護を解除する **unprotect** は、シートとブックの両方に対して使用できます。

```
unprotect (active sheet) password "drowssap"
```

Dictionary には、このコマンドのダイレクト パラメータが列挙型定数の書式 (**sheet** / **workbook**) で記載されていますが、実際は、必要なオブジェクトのクラス (**sheet** または **workbook**) を示します。

同じ **password** を使用してブック内の全シートを簡単に保護するには、次のようにします。

```
Const sPWORD As String = "drowssap"
Dim ws As Worksheet
For Each ws In
ActiveWorkbook.Worksheets
        ws.Protect
Password:=sPWORD
Next ws
```

今度も、ループの使用を回避できます。

```
set sPWORD to "drowssap"
protect worksheet (every worksheet in active workbook) password sPWORD
```

複数のアイテムに対して実行できるのは、**open**、**delete** などの Standard Suite コマンドだけではありません。**protect worksheet**、**unprotect** など、Microsoft Excel Dictionary にあるその他多数のコマンドも同様に利用できます。Excel 独自のコマンドの多くは明らかに、複数アイテムに対して実行できるように実装されていますが、そうではないコマンドもあるため、初めて使用するコマンドは 1 つずつ確認する必要があります。

ワークシートを非表示にする

ユーザーから見えないようにワークシートを非表示にすることができます。VBA で全ワークシートを非表示にするには、それぞれの名前に (hide) を使用し、次のように記述します。

```
Const sHIDEINDICATOR As String = "(Hide)"
Dim ws As Worksheet
For Each ws In Worksheets
        ws.Visible = Not (LCase(ws.Name) Like _
                LCase("*" & sHIDEINDICATOR & "*"))
Next ws
```

AppleScript では、次のように記述します。

```
tell application "Microsoft Excel"
        repeat with ws in (get every worksheet in active workbook)
                set visible of ws to not (name of ws contains "(Hide)")
        end repeat
end tell
```

AppleScript の文字列演算子は、text item delimiters を除き、considering case ブロック内に記述しない限り大文字小文字を区別しません。このため、任意の語を "(Hide)" または (hide) と比較する際に、VBA の例のような追加コードを使用する必要がありません。

スクリプティング機能追加がインストールされていない標準の AppleScript にはワイルドカードまたは正規表現がない代わりに contains が存在し、これがきわめて有効です。

次に、every worksheet whose... が機能しない例を紹介します。これは、Excel がこのような構造のプロパティ (**visible** など) を処理できないためです。

```
set visible of (every worksheet in active workbook ¬
        whose its name contains "(Hide)") to false
--> ERROR: Can't set visible of every worksheet of active
workbook whose name contains "(Hide)" to false.
```

ここでは、repeat ループを使用する必要があります。

ただし少し変更が加えられているので注意が必要です。上の VBA コードでは列挙 *xlHidden* の同義語として、*Not* (ws.Visible = False の意味) を使用しました。ワークシートが [再表示] ダイアログ ボックス ([書式] メニューの [シート] をポイントし、[再表示] をクリック) にも表示されないようにするには、代わりに **Visible** プロパティを *xlVeryHidden* に設定します。その場合は次のようになります。

```
ActiveSheet.Visible = xlVeryHidden
```

AppleScript では次のようになります。

```
set visible of active sheet to sheet very hidden
```

AppleScript では **visible** プロパティの適切な列挙は、*sheet hidden*、*sheet very hidden*、および *sheet visible* であり、これらはそれぞれ VBA の *xlHidden*、*xlVeryHidden*、および *xlVisible* に相当します。必要な場合は VBA と同じく、*sheet visible* と *sheet hidden* の同義語として、それぞれ *true* と *false* を使用できます。

列挙定数の実際の Long (数値) を確認する場合は、Microsoft Visual Basic Editor のオブジェクト ブラウザを使用する方が多少わかりやすいですが、AppleScript でも確認することはできます。

これは、これらの VBA から AppleScript への変換を行う際に便利なものです。VBA マクロで定数 *xl* または *vb* の列挙ではなく数値が使用されている場合は、AppleScript Dictionary に参照先の列挙定数が明示的に定義されていなかったり、定数にこのような "相当する数値" が存在したりしても、一般に同じ数値を AppleScript コードに転記できます。

行と列を使用する

Excel 2004 の既知の問題

Microsoft Excel 2004 for Mac には、AppleScript での行と列の取り扱いに関する問題がいくつかあります。次の例はすべて、便利で機能的な **range** を生成します。結果は **column** または **row** として参照されることはなく、常に **range** として参照されます。

```
row 5 of (get used range of active sheet)
column 7 of range "A1:P26" of active sheet
range "G1:G26" of range "A1:P26" of active sheet
```

一方、**range** としてではなく、**column** として参照すると、結果は生成されません。

```
column "G1:G26" of range "A1:P26" of active sheet
```

したがって、列と行はアドレスによってではなくインデックス (数値) によって参照します。

通常、VBA でも `Columns(7)` や `Rows(5)` を参照します。Dictionary では、**column**、**row**、および **cell** はすべて **range** のサブクラスであり、そのプロパティをすべて継承すると記載されています。つまり、**column**、**row**、および **cell** では、**range** と同じことを実行できると思われますが、実際はそうではありません。ほとんどの場合、列と行は必ずインデックスによって参照し、アドレスを使用する場合は **range** を使用します。

また、次のスクリプトでも何も生成されません。

```
rows of (get used range of active sheet)
columns of range "A1:P26" of active sheet
```

実行すると、予期していたアイテムのリストではなく、単一の行または列のオブジェクトだけが取得されます。これはリストとして使用できず、個々のアイテムを取得することはできません。取得しようとするとスクリプト エラーになります。

また、次のような `repeat` ループは実行することができません。

```
set theRows to rows of (get used range of active sheet)
repeat with theRow in theRows
        set v to value of theRow
end repeat
```

このスクリプトは、その存在を無視するかのようにループを "ジャンプ" します (デバッガ エディタで 1 行ずつ実行するとこれを確認できます)。そして、ループの実行内容が設定された結果や変数を使用しようとした時点でエラーになります。

ただし、**count** (**every row**) または **count rows** (同じコマンドです) は有効なので、別の方法で目的を達成できます。**rows** を取得することはできませんが、(**count rows**) を取得することができます。したがって、`repeat` ループを繰り返して、`row i` を取得できます (ここで、`i` は `1 to (count rows)` でループします)。

また、AppleScript では、参照中に行や列に変数を設定することができません。これも結果が得られないためです。行や列への参照は、変数に評価されると、常に実際の行や列と関係のないオブジェクトになります。変数を使用しないアプリケーション参照によって各要素を順に指定することによってのみ正常に動作します。

したがって、次のように記述する必要があります。

```
tell (get used range of active sheet)
        repeat with i from 1 to count rows
                set v to value of row i
        end repeat
end tell
```

必要な最終結果を得られるまで待ってから、変数をその最終結果 (この例では **value** プロパティ) に設定してください。参照中の行、列、セルには設定しないでください。変数を **range** として参照されるものに設定することは、まったく問題ありません。これには、**used range**、アドレス指定による範囲 (`range "A1:A46"`)、実際は行や列である範囲なども含まれます。しかし、`row 5` や `column 8` などのように定義されたものに変数を設定することはできません。

したがって、次のように変数 `ur` が範囲に設定されているコードは正しく動作します。

```
set ur to (get used range of active sheet)
tell ur
        repeat with i from 1 to count rows
                set v to value of row i
        end repeat
end tell
```

しかし、変数 `theRow` が行に設定されていると動作しません。

```
set ur to (get used range of active sheet)
tell ur
        repeat with i from 1 to count rows
                set theRow to row i
                set v to value of theRow
        end repeat
end tell
```

これらの "法則"、つまり動作は発見しにくいため、このトピックをいつでも参照できるようにしておいてください。

また、AppleScript リストに解決されない `every column of someRange` や `rows of someRange` などは、先に説明した、結果を返さないオブジェクトであり、それらに対してコマンドを実行することはできません。したがって、`repeat` ループで、行や列の変数を使用せずに単一アイテムに一度に 1 つずつアクセスするという、前に述べた信頼できる方法のみを使用する必要があります。

この動作の原因として考えられるのは、VBA コレクション オブジェクトです。行と列をカウントでき、アドレスではなくインデックスで参照できることは、VBA スクリプト作成者にとって慣れ親しんでいる状況です (VBA では **Rows Collection** オブジェクトと **Columns Collection** オブジェクトに 2 つのメソッドを使用できます)。複数形を個々のアイテムのリストに解決できないことも同様です。

セルを `every cell of someRange` で使用しようとする場合にも、同様の問題があります。ただし、セルは範囲として返されるので、変数を設定することはできますが、できるだけ **range** を使用してください。

値、高さ、幅を選択して設定する

AppleScript では、個々の **cell** だけでなく、**row** 全体、**column** 全体、または任意の **range** 全体についても AppleScript リストを使用することで、値を取得したり設定することができます。範囲の値はリストのリストとして表され、内側のリストが行、内側のリストの項目がセルになります。たとえば次のようになります。

```
{{1, 2, 3}, {10, 20, 30}, {100, 200, 300}, {1000, 2000, 3000}}
```

これは 3 列 4 行の範囲を表し、一番上の行が `{1, 2, 3}` です。

1 行だけで構成される範囲は、単一のリスト `{1, 2, 3}` と考えることができ、3 列を含む行の値をこのように設定する方法は有効です。

一方、Microsoft Excel から返される値は、常にリストのリスト `{{1, 2, 3}}` となります。**row** 自体が **range**、つまり 1 行の範囲と見なされ、これが正しい表現として使用されます。このように設定する方法も有効です。

列の値は常に `{{1}, {10}, {100}, {1000}}` のように表されます。これは内側のリストの項目が 1 つずつしかない範囲と考えることができます。

単一セルの値は `1` や `10` のように値のみとして返され、リストのリストとしては返されません。便宜上このように設定されていますが、このように設定することが一般的です。

ただしセルの値を `10` ではなく `{{10}}` のように設定することもできます。セルに表示される値は `10` と変わりません。

このようにリストのリストとして **row**、**column**、または **range** から取得、あるいはこれらに設定できるプロパティは、**value** プロパティだけです。**string value** プロパティであっても、複数のセルのリストには対応していません。

以上のことを念頭に置いて、Visual Basic for Applications (VBA) の簡単なプロシージャをいくつか紹介します。

アクティブ セルの行または列を選択する

VBA では、**.EntireRow** または **.EntireColumn** プロパティを使用して、アクティブ セルの行または列を選択します。

```
ActiveCell.EntireRow.Select
```

AppleScript では次のようになります。

```
select entire column of active cell
```

(**select** は Standard Suite にあります。) これは完全に機能します。

行の高さ、列の幅、自動サイズ調整

行の高さは、**row height** プロパティを使用してポイント単位で指定します。一方、列の幅は、**column width** プロパティを使用して標準の数字で指定します (幅がゼロの場合は 0)。これは [環境設定] で設定されている標準のフォントとサイズによって変わります。VBA では、列と行に対して **Autofit** コマンドを実行し、内容が入る最小限の高さと幅にサイズを縮小することもできます。

しかし、AppleScript の **autofit** コマンドには問題があります。これら 3 つをすべて使用した、次のようなマクロがあるとします。

```
Range("1:4").RowHeight = 20  'points
Range("A:C").ColumnWidth = 20  'characters
With Range("J5:Z43")
        .Columns.AutoFit
        .Rows.AutoFit
```

最初の 2 つのステートメントはそのまま AppleScript に変換できます (この AppleScript は後に示します)。行の高さと列の幅を設定することについては問題ありません。

autofit コマンドを使用する場合は、列ごとにループを実行する必要がありますが、このコマンドを使いこなせれば非常に便利です。**columns** と複数形で指定したり、**every column** と指定して自動サイズ調整を実行しようとしても、何も起こりません。

列の場合はこのように対処できますが、行の場合は 1 行だけであったとしても **autofit** コマンドが動作しません。実行すると、行の高さを狭める代わりにいくつかの列の幅が狭まり、それらに適用済みの自動サイズ調整も無効になってしまいますが、行に対しては何も起こりません。実際に行の高さを変更することはほとんどないので、変更する場合は個々に設定する必要があります。

上に示したマクロは、次のように変換する必要があります。

```
tell application "Microsoft Excel"
        set row height of range "1:4" to 20 -- points (rows)
        set column width of range "A:C" to 20 -- characters (columns)
        tell range "J5:Z43" of active sheet
                repeat with i from 1 to (count columns)
                        autofit column i
                end repeat
                -- now see if here are any rows whose row height
                -- has to be adjusted, and do them separately
        end tell
end tell
```

範囲を使用する

範囲の基本

AppleScript の実装では、範囲に影響する可能性のある問題がいくつかあります。これは、スクリプトにおいて **range** としてではなく **row**、**column**、または **cell** として範囲を参照した場合の問題です。列と行に関する問題の詳細については、「Excel 2004 の既知の問題」を参照してください。

範囲は、Visual Basic for Applications (VBA) および AppleScript を使用して Microsoft Excel を操作する際の基本となります。範囲は **worksheet** の子オブジェクトであり、複数のシートのセルを含めることはできません。

AppleScript では、範囲を `A1` 形式のアドレスで参照します (他のアプリケーションでは名前で参照します)。つまり、キーワード **range** の後にスペースを 1 つ空けて、引用符で囲んだアドレスを指定します。VBA のようにかっこは使用しません。

- `Range("A1:J10")`
- `range "A1:J10"`

ここで注意が必要な問題が 1 つあります。VBA と AppleScript を使用しているほとんどのユーザーが気付いていますが、**sheet** を指定せずに、VBA で `Range("A1:J10")`、AppleScript で `range "A1:J10"` または `cell "A1"` と書くと、Excel の既定で **active sheet** が選択されます。

しかし、**used range** を参照して次のように指定する場合があるとします。

```
set ur to used range
```

または

```
cell 3 of row 4 of (get used range)
```

このときに `of active sheet` を指定しなかった場合、結果は返されずに、結果を参照するスクリプトの次の行でエラーになります。

どのような場合でも、常にシートを指定することをお勧めします。たとえば `tell active sheet` を使用して、スクリプト内の必要な部分を囲みます (VBA で `With ActiveSheet` を使用する場合と同じです)。多くのスクリプトに言えることですが、**active sheet** プロパティを省略することが習慣になっていると、**used range** プロパティにアクセスする必要があるときにエラーが発生して困惑することになります。したがって、省略することを習慣化しないことが重要です。

値を選択して設定する

Visual Basic for Applications (VBA) で範囲を指定するには、**Worksheet** (または別の Range) に適用した **Range** プロパティ、あるいは Range に適用した、Range の **Cells** プロパティを使用します。**Cells** プロパティは **Range** オブジェクトの既定のプロパティであるため、通常は名前を指定して明示的に呼び出すことはありません。**cell** の **row** と **column** を表す 2 つの数字がかっこ ( ) で囲まれているだけの場合もあります。

AppleScript には、**range** または **worksheet** の "cells" プロパティはありません。AppleScript では、単に適切な数字を使用して、**row** (行番号) の **cell** (列番号) を取得します。次の AppleScript の 2 行目がこれに該当します。つまり、VBA では `.Cells(10, 6)` または単に `.(10, 6)` と記述するところを、AppleScript では `cell 6 of row 10` と記述します。

範囲は直接選択して設定できます。

```
Range("A1").Select
ActiveWorksheet.Cells(10, 6).Select
Set rMyRange = Range("B10:C14")
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        select range "A1"
        select (cell 6 of row 10 of active sheet)
        set myRange to range "B10:C14"
end tell
```

範囲を拡張したり選択するには、Table Suite の **get resize** コマンドを使用できます (Table Suite には、範囲に使用できる多くのコマンドあります)。VBA では **.Resize** プロパティ (またはメソッド) を使用します。

```
Range("A1").Resize(10, 10).Select 'Selects A1:J10
```

AppleScript では次のようになります。

```
select (get resize range "A1" row size 10 column size 10)
--selects A1:J10
```

連続していない範囲を単一の値に設定する

範囲の値をリスト (行) の一覧に設定する方法については、「値、高さ、幅を選択して設定する」を参照してください。VBA では次のような短縮パターンも多く使用できます。

```
Range("A1:B5,G9,A16:D19") = 5
```

この Range アドレスでは、A1:B5 内に 10 個のセル (5 行の 2 列) を含む矩形、単独セル G9、および 16 個のセル (4 行の 4 列) を含む孤立した矩形 A16:D19 から成る連続していない範囲を指定しています。このステートメントは、範囲内の各セルの値を 5 に設定するという命令を簡略化したものです (実際にこのような範囲を整数 5 に設定することはできません)。このステートメントは有効です。

これが有効なのは、VBA には既定のプロパティがあり、明示的な記述を省略できるという理由もあります。**Cells** は **Range** の既定のプロパティであり、**Value** は **Cell** の既定のプロパティです。

一見すると、これを AppleScript では再現できそうにありませんが、Microsoft Excel AppleScript に既定のプロパティがないことを踏まえて、次のようにプロパティを明示的に記述するのであれば再現できます。

```
set value of every cell of range "A1:B5,G9,A16:D19" to 5
```

これは純粋な AppleScript であり、簡略化する必要はありません。このコードでは、VBA のコードと同様に、連続していない範囲の各セルの値が一度に 5 に設定されます。

コピー、ペースト、および形式を選択してペーストする

range をある場所から別の場所へコピーする最も簡単な方法は、**copy range** コマンドを使用することです。これは、Visual Basic for Applications (VBA) の **Copy** メソッドに相当します。

```
With ActiveSheet
        .Range("A1:J10").Copy Destination:=.Range("M1")
End With
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        copy range range "A1:J10" destination range "M1"
end tell
```

range などのキーワードは、コマンド、ダイレクト オブジェクト、およびパラメータで共通して使用されます。このため、構文の中では同じ単語が 2 回繰り返されています。VBA と同様に AppleScript でも、**destination range** には左上の最初の **cell** だけを指定します。

また、VBA と同様に of active sheet が既定として使用されます。

copy range コマンドでは、値、書式、条件付き書式、入力規則などをコピーすることができます。

クリップボードの内容をアクティブ セルにペーストするには、(ここでは Microsoft Excel Suite の) **paste worksheet** コマンドを使用します。このコマンドは、特定のセルにペーストする VBA の **Paste** メソッドに相当するものです。オプションの引数 **destination** を使用することもできます。

```
ActiveWorkbook.Sheets("Sheet1").Range("A1:J10").Copy
With Workbooks("Book2.xls").Sheets("Sheet1")
        .Paste Destination:=.Range("J3")
End With
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        copy range (range "A1:J10" of sheet "Sheet1" of active workbook)
        tell sheet "Sheet1" of workbook "Book2.xls"
                paste worksheet destination range "J3"
        end tell
end tell
```

値だけをコピーする場合は、**copy range** コマンド (VBA では **Copy** メソッド) を呼び出すよりも、単純に値を割り当てた方がより効率的にすばやく実行できます。

```
With Sheets("Sheet1").Range("A1:J10")
        Sheets("Sheet2").Range("B10").Resize(_
                .Rows.Count, .Columns.Count).Value = .Value
End With
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        tell (range "A1:J10" of sheet "Sheet1")
                set value of (get resize (range "B10" of sheet ¬
                        "Sheet2" of active workbook)
                        row size (count rows) ¬
                        column size (count columns)) ¬
                        to get its value
        end tell
end tell
```

この場合、`Sheet2` には `of active workbook` を指定する必要があります。この理由は、同じ `tell` ブロック内で別の **sheet** である `Sheet1` の範囲がターゲットに設定されているためです。`of active workbook` を指定しなかった場合、`range A1:J10` の中で `Sheet2` が検索されることになり、この範囲内にシートが見つからなければ何も実行されません。

同様に、同じコマンドで `its value` が評価されるようにするには、**get** を明示的に指定する必要があります。ただし前の行で変数をこの値に設定するのであれば、**get** は必要ありません。実際、Microsoft Excel では、Microsoft Word のように **get** を明示的に指定する必要はありませんが、この場合は特別です。`its` は、このコードでは省略可能ですが、対象となる **value** の所属先 (元の範囲) を識別するのに役立ちます。また、`tell` ブロックでは、`it` が必要になることが数多くあります。

形式を選択してペーストする

Table Suite の **paste special** コマンドは、VBA の **PasteSpecial** メソッドに対応します (Excel Suite には **paste special on worksheet** というコマンドがありますが、これは範囲に関係のない別のコマンドです)。このコマンドを使用すると、ペースト先の範囲に対し、クリップボードの内容を基に加算、減算、乗算、除算などの操作を実行できます。

たとえば、範囲内の各セルの値に 2 をかけるには、次のようにします。

```
Application.ScreenUpdating = False
' Get the value in the clipboard using a
' cell in a temporary worksheet
With ActiveWorkbook.Worksheets.Add
        With .Cells(1, 1)
                .Value = 2
                .Copy
        End With
End With
Sheets("Sheet1").Range("A1:J10").PasteSpecial_
                Paste:=xlPasteAll, _
                Operation:=xlPasteSpecialOperationMultiply
' Delete the temporary worksheet
With Application
        .DisplayAlerts = False
        ActiveSheet.Delete
        .DisplayAlerts = True
        .ScreenUpdating = True
End With
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        set screen updating to false -- speed it up
        --get the value in the clipboard to a cell
        --in a temporary worksheet
        tell (make new worksheet at active workbook)
                tell cell 1 of row 1
                        set value to 2
                        copy range -- to clipboard
                end tell
        end tell
        paste special (range "A1:J10" of sheet "Sheet1") ¬
                what paste all operation paste special operation ¬
                multiply
        --delete the temporary worksheet
        set display alerts to false
        delete active sheet
        set display alerts to true
        set screen updating to true
end tell
```

Dictionary で **paste special** とそのパラメータのさまざまな列挙型を確認してください。**Paste** ではなく **what** となっているように、キーワードの名前が VBA と異なっていることがあります。これは、キーワードの範囲が幅広い AppleScript での用語の重複を避けるためです。スクリプティング機能追加では `paste` は非常に一般的な用語です。

範囲を変換する

paste special コマンドは、列と行を変換 (**transpose**) する場合にも使用できます。ペースト先の範囲に複数のセルがある場合、その範囲の行数と列数はペースト元の範囲と同じである必要があります。

```
With ActiveSheet
        .Range("A1:E2").Copy
        .Range("A4:B8").PasteSpecial
Transpose:=True
End With
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        copy range range "A1:E2"
        paste special range "A4:B8" with transpose
end tell
```

paste special コマンドでは、矩形の範囲を使用していたとしても、現在の範囲を変換することはできません。また、**transpose** *true* をコンパイルすると、*with* **transpose** にコンパイルされることにも注意が必要です。

空白セルを挿入する

insert into range コマンドを使用して **range** に空白のセルを挿入することができます。このコマンドは、Visual Basic for Applications (VBA) の **Insert** メソッドに相当し、UI の [挿入] メニューの [行] 、[列] 、または [セル] をクリックするのとまったく同じです。VBA では次のようになります。

```
Sheets("Sheet1").Range("B2:B5").Insert Shift:=xlShiftToRight
```

AppleScript では次のようになります。

```
insert into range range "B2:B5" shift shift to right
```

VBA では、ある範囲をコピーしてそれと同じサイズの範囲に **Insert** メソッドを適用する場合、クリップボードの内容が挿入先のセルにコピーされます。

```
With ActiveWorkbook
        With Sheets("Sheet2").Range("A1:A10")
        .Copy
        Sheets("Sheet1").Range("B2").Resize( _
                .rows.count, .columns.count).Insert _
                Shift:=xlShiftToRight, _
                CopyOrigin:=xlFormatFromLeftOrAbove
        End With
End With
```

AppleScript にはこの VBA の引数 **CopyOrigin** に相当するものがないため、既定の設定が適用されます。この引数を省略すると、移動方向が右方向であれば左側のセルの書式が、下方向であれば上側のセルの書式が継承されます。AppleScript にはこの引数を指定する方法がありませんが、この例では *xlFormatFromLeftOrAbove* を指定するのとまったく同じ結果が得られます。

```
tell application "Microsoft Excel"
        tell active workbook
                tell range "A1:A10" of sheet "Sheet2"
                        copy range
                        insert into range (get resize row size (count rows) ¬
                                column size (count columns)) shift ¬
                                shift to right
                end tell
        end tell
end tell
```

範囲を削除する

range を削除すると、引数の指定内容に応じてセルが上または左に移動します。

次の例では、最初の列と同じ値を持つ、重複した行を削除します。Microsoft Excel では、行を削除すると行番号が付け直されるため、ループは `Step -1` を使用することにより、**row** の番号が最大の行から最小の行に向かって "逆方向" に実行されます。これは AppleScript 自体のインデックス作成時の動作と同じであり、AppleScript では必然的にこの方法を使用することになります (`repeat with i from lastRowNum to 1 by -1`)。Visual Basic for Applications (VBA) では、次のようになります。

```
Dim i As Long
With ActiveSheet
        For i = .Cells(.Rows.Count, 1).End(xlUp).Row To 2 Step -1
                If .Cells(i, 1).Value = .Cells(i - 1, 1).Value Then _
                        .Cells(i, 1).EntireRow.Delete
        Next i
End With
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        tell active sheet
                set lastRowNum to first row index of ¬
                        (get end (cell 1 of row (count rows)) ¬
                                direction toward the top)
                repeat with i from lastRowNum to 2 by -1
                        if value of (cell 1 of row i) = ¬
                                value of (cell 1 of row (i - 1)) then
                                delete entire row of (cell 1 of row i)
                        end if
                end repeat
        end tell
end tell
```

AppleScript では、`repeat` ループ ステートメント内で行番号の評価を行おうとするとコードがきわめて複雑になります (VBA ではこれより短い用語とドット構文を使用します)。したがって、これは変数として処理する方が適切です。「Excel 2004 の既知の問題」で説明しているように、行には変数を使用しないでアプリケーションの用語を直接使用することが基本ですが、範囲や、この例でも使用している数値 (あるいは文字列、日付、またはその他の基本的な型) に変数を設定することもできます。

VBA に長いドット構文による参照がある場合は、AppleScript に変換する際、最後から順に逆方向に指定するのが自然です。ただし、プロパティの場所を見つけるときは、プロパティが属する上位の VBA オブジェクトから開始します。VBA では、プロパティを指定する同じドット構文と親子関係の中にメソッド名が出現することがあります。一方、メソッドに対応する AppleScript のコマンドは、プロパティまたは要素と同じ `of` 構文を使用することはありません。このコマンドは、適切な Dictionary スイートのコマンド セクションで探し出す必要があります。

VBA のプロパティは、AppleScript では要素の場合もあれば、要素のプロパティ、またはコマンドの場合もあります。このため、すべてを特定して書き換えるには時間がかかります。

`Cells(r, c)` 構文は次の AppleScript に変換されます。結果は範囲です。

```
cell c of row r
```

たとえば、**End** プロパティに対応するものを、Table Suite の **range** クラスで探します。これは **end of range** のようなものと考えられますが、**range** クラスにはそうしたものが見つかりません。

そこで、Table Suite 内のコマンドに関するセクションで、範囲の最後を取得できそうなものを探します。`end of...` 以下を見ると何もありません。次は、近くの **get** を調べます。AppleScript でプロパティに対応するものがなく専用コマンドが必要になる場合、そのコマンド用語は **get** で始まっていることがよくあるため、この方法をお勧めします。

そうすると、**get end** が見つかります。このコマンドはダイレクト オブジェクトまたはパラメータとして **range** を取得します (特定のパラメータ キーワードが必要ないため、その意味で "ダイレクト" と呼ぶことができます)。また、VBA の *xlTop* に対応する *toward the top* を指定することにより、この **direction** パラメータは、必要な列挙型を取得します。この意味は "先頭から開始して最後のもの" であり、先頭を中心として方向を定めています。もちろん、今探しているのは最後の行です。

ある範囲の **rows** または **every row**、あるいは **sheet** を取得することはできませんが、`get count rows` (または `count every row`) を使用することができます。最後に、**Range** メソッドの **Row** プロパティは、AppleScript では **first row index** プロパティになります。これが今必要としている、整数値を返す説明的な用語です (プロパティ `row` はクラス **row** と競合するため、AppleScript はこのプロパティを呼び出しません)。

これで、**cell** の行番号を返すステートメントの作成に必要なものが全部揃いました。

```
set lastRowNum to first row index of ¬
        (get end (cell 1 of row (count rows)) direction toward the top)
```

次は、AppleScript で `tell` ブロックのターゲットであるシート自体に対して **get end** コマンドを実行したときに、`65535` ではなく最後に使用されている行がシートの最終行になるかどうかを確認します。最後に使用されている行ではない場合、シートの **used range** をターゲットとして明示的に指定する必要があるかどうかも確認してください。AppleScript では、VBA の **Sheet.End** と同様に最終行を強制的に指定でき、`65535` ではなく最後に使用されている行の番号を取得できます。

スクリプトの残りの部分では、`cell c of row i` をさらに何度か使用して、それぞれの範囲 (セル) の **value** プロパティを取得します。`repeat` ループが `by -1` (VBA の `Step -1` に相当) によって逆方向に進むようにしておけば、`delete entire row` ステートメントは正しく実行されます。

重複する行をすべて 1 つの範囲変数に格納し、**delete** コマンドで一度に消去するとさらに効率的です。VBA の **Union** メソッドには両方の **Range** オブジェクトを設定する必要がありますが、AppleScript の **union** コマンドも同じであることに注意してください。

```
Dim rCell As Range
Dim rDelete As Range
With ActiveSheet
        For Each rCell In .Range("A2:A" & _
                .Range("A" & .Rows.Count).End(xlUp).Row)
                If rCell.Value = rCell.Offset(-1, 0).Value Then
                        If rDelete Is Nothing Then
                                Set rDelete = rCell
                        Else
                                Set rDelete = Application.Union(rDelete, rCell)
                        End If
                End If
        Next rCell
        If Not rDelete Is Nothing Then _
                rDelete.EntireRow.Delete
End With
```

AppleScript バージョンを次に示します。

```
tell application "Microsoft Excel"
        tell active sheet
                set lastRowNum to first row index of ¬
                        (get end (cell 1 of row (count rows)) ¬
                                direction toward the top)
                set rDelete to missing value
                repeat with i from 1 to (count cells ¬
                        of range ("A2:A" & lastRowNum))
                        --that's column A from 2nd cell to bottom used cell
                        set rCell to (cell i of range ("A2:A" & lastRowNum))
                        if value of rCell = value of ¬
                                (get offset rCell row offset -1) then
                                if rDelete = missing value then
                                        set rDelete to rCell
                                else
                                        set rDelete to union ¬
                                                range1 rDelete range2 rCell
                                end if
```

```
                    end if
                end repeat
                if rDelete  missing value then delete entire row of rDelete
        end tell
end tell
```

ここでは、最終行の番号を変数として設定し、`repeat` ステートメントの処理を簡単にしています。AppleScript では、未定義の変数は `null` (ごくまれに使用される `Nothing`) と同等ではありません。未定義の変数を呼び出すとエラーが発生します。通常、このようなエラーは `try/on error` ブロックでトラップしますが、この場合は同じ `try` ブロック内に **union** ステートメントも指定するため、別のエラーがキャッチされると考えられます。そうすると、`rDelete` を最初のセルに設定し続けるという無限ループに陥る可能性があります。または、複雑なエラー制御を行ったり指定順を変更したりすることになり、VBA との比較がさらに難しくなります。

そこで、`rDelete` を独自の値に初期化する方法が役に立ちます。この値は、Excel の範囲でなければ何でもかまいません。初期値として一般的なのは `""` や `0` です。一方、未定義でもエラーが発生しないものに対しては AppleScript の便利な式である `missing value` を使用することもできます (これは、スクリプト作成者または Excel のスクリプト記述ミスがあった場合によく見られます)。これは厳密には `null` とは違いますが、ここでの目的は達成できます。

同じように、(every cell of someRange) を取得したり、これに変数を設定することはできませんが、count cells of someRange (!) を実行することができます。repeat ループでは、repeat with rCell in (cells of someRange) 構文を使用しないようにする必要があります。これによって rCell を item i of (cells of someRange) に設定することも回避されます (この場合の someRange は range ("A2:A" & lastRowNum) 全体)。

代わりに、次を参照するようにします。

```
cell i of someRange
```

VBA の **Offset** メソッドは **get offset** によって再現できます。パラメータ名の **row offset** と **column offset** を記述する必要がありますが、それぞれの既定値は 0 であるため、**column offset** は省略できます。

Table Suite ではなく Microsoft Excel Suite にある **union** コマンドは、範囲に対してのみ作用し、**range1**、**range2** などのパラメータを使用します。また、`delete entire row` は連続していない範囲に対して実行できます。

ただし、**delete** コマンドでは範囲リストを一度に消去できるので、AppleScript では VBA にはない別の方法を使用することもできます。

```
tell application "Microsoft Excel"
        tell active sheet
                set lastRowNum to first row index of ¬
                        (get end (cell 1 of row (count rows)) ¬
                                direction toward the top)
                set rDelete to {}
                repeat with i from 1 to (count cells ¬
                        of range ("A2:A" & lastRowNum))
                        --that's column A from 2nd cell
                        --to bottom used cell
```

```
                        set rCell to (cell i ¬
                                of range ("A2:A" & lastRowNum))
                        if value of rCell = value of ¬
                                (get offset rCell row offset -1) then
                                set end of rDelete to (entire row of rCell)
                        end if
                end repeat
                if rDelete  {} then delete rDelete
        end tell
end tell
```

こちらの方が簡単です。`rDelete` は、空のリスト `{}` に初期化します。重複するセルを検出した場合は、リスト `rDelete` の最後にそのセルの **entire row** を追加します。このとき、リストが空白かどうかを確認する必要はありません。これで、最後にリストを削除して完了です。

VBA を使い慣れていると、マクロの変換を始める際にそれを正確に反映しようと思いがちですが、AppleScript に慣れてくるに従い、この言語の特長を生かしてより自然なスクリプトを記述できるようになります。

範囲を並べ替えてフィルタを設定する

データを並べ替えるには、Visual Basic for Applications (VBA) では **Sort** メソッド、AppleScript ではこれに相当する **sort** コマンドを使用します。1 つの **cell** が指定されている場合、**current region** (空白の行と空白の列で囲まれたセル領域) が並べ替えられます。次のコードの抜粋では、最初に `column A` の値、次に `column B` の値に基づいて、`cell A1` の周りのデータ領域が並べ替えられます。

```
With ActiveSheet
        Range("A1").Sort _
                Order1:=xlAscending, _
                Key1:=.Columns("A"), _
                Order2:=xlAscending, _
                Key2:=.Columns("B"), _
                Header:=xlNo, _
                MatchCase:=False
End With
```

これは AppleScript でもまったく同じです。**match case** *true* が *with* **match case** にコンパイルされます。

```
tell application "Microsoft Excel"
        tell active sheet
                sort range "A1" order1 sort ascending key1 column 1 ¬
                        order2 sort ascending key2 column 2 header ¬
                        header no ¬
                        without match case
        end tell
end tell
```

AppleScript では column A を参照できず、スクリプトはエラーになります。指定できるのは、column 1、column "A:A"、または range "A:A" です。

```
sort range "A1" order1 sort ascending key1 range ¬
        "A:A" order2 sort ascending key2 range "B:B" header header no ¬
        without match case
```

オートフィルタを使用して範囲にフィルタを設定する

各シートには **Autofilter** オブジェクトが含まれ、個々のオートフィルタは **AutoFilter** メソッドを使用して適用されます。各フィルタには 2 つの条件を設定でき、フィルタは累積して適用されます。次の例では、column A の値が >=10 および <=100、column C の値が OK である行のみを **sheet** に表示します。

```
With ActiveSheet.Range("A1")
        .AutoFilter 'remove existing autofilter dropdowns
        .AutoFilter _
                Field:=1, _
                Criteria1:=">=10", _
                Operator:=xlAnd, _
                Criteria2:="<=100", _
                VisibleDropdown:=True
        .AutoFilter _
                Field:=3, _
                Criteria1:="OK", _
                VisibleDropdown:=True
End With
```

1 つのセルのみが指定されていることから、オートフィルタは Cell A1 の周りの **CurrentRegion** に適用されます。

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        tell range "A1" of active sheet
        --single cell, applies to current region
                autofilter range
                -- with no parameters, toggles dropdowns
                autofilter range field 1 criteria1 ">=10" operator ¬
                        autofilter and criteria2 "<=100" ¬
                        with visible drop down
                autofilter range field 3 criteria1 "OK" ¬
                        with visible drop down
        end tell
end tell
```

ここで難しいことは、正しい **autofilter** 用語を探すことだけです。マクロ内の **.AutoFilter** は、**worksheet** のプロパティではなく、シートの **AutoFilter** オブジェクトを返すものではありません。これは **Range** のメソッドです。このことを理解するには、VBA に関する十分な知識が必要です。したがって、Microsoft Excel Suite の **sheet** の **autofilter** クラスや **autofilter object** プロパティは気にしなくてもかまいません。クラスがターゲットとなる Table Suite で **range** を探すと、メソッドに対応する **autofilter range** コマンドが見つかります。他の部分はすべて同じです。

最後のセルを検索する

新しい記録を開始する際など、**range** 内で最後に使用したセルを検索する必要がある場合がよくあります。これを行う方法の 1 つに、**End** メソッドを使用した次のような方法があります。

```
Dim rNext As Range
With Sheets("Sheet1")
        If IsEmpty(.Range("A1").Value) Then
                Set rNext = .Range("A1")
        Else
                Set rNext = .Range("A" & .Range("A" & _
                        .Rows.Count).End(xlUp).Row).Offset(1, 0)
        End If
End With
```

ドットとかっこがネストされているので、AppleScript への単純変換は容易ではありません。そのまま AppleScript に書き換えると次のようになります。

```
tell application "Microsoft Excel"
        tell sheet "Sheet1" of active workbook
                if value of range "A1" = "" then
                        set rNext to range "A1"
                else
                        set rNext to get offset of range ("A" & ¬
                                (first row index of (get end range ¬
                                ("A" & (count rows)) ¬
                                        direction toward the top))) ¬
                                        row offset 1
                end if
        end tell
end tell
```

少なくとも、よく出現する構造について十分理解できるようになるまでは `ofs` を一度に多く使用せず、途中に中間変数を介在させることをお勧めします。たとえば次のようにします。

```
set lastRowCellA to (get end range ¬
        ("A" & (count rows)) direction toward the top)
        set lastRowNum to (first row index of lastRowCellA)
        set lastRowCellAA to range ("A" & lastRowNum)
        set rNext to get offset of lastRowCellAA row offset 1
```

実際、このように書き出してみると、冗長で回りくどい構造になっていることがわかります。1 行目で `lastRowCellA` を設定すると **get offset** をすぐに使用できます。**row index** を取得してアドレスに戻す必要はありません。

```
set lastRowCellA to (get end range ¬
        ("A" & (count rows)) direction toward the top)
set rNext to get offset of lastRowCellA row offset 1
```

これもまったく同じことを行っています。このように簡単になると、変数を取り除くこともできます。しかし、あまりにも複雑な場合は、コードを分割してコメント的な変数名を使用しておくのが一般的であり、コードがよりわかりやすくなって便利です。実際、この Visual Basic for Applications (VBA) コードは、単一セルではなくより広い範囲を取得するために記述されたものです。そのような場合は手順を追加する必要があります。

変数は (名前付きの) **column**、**row**、または **cell** に設定しないで、**range** に設定する必要があります。つまり、`A1` のような形式のアドレスを使用して、`column` を `range` に置き換えるだけです。

値のオートフィルを実行するまたは値を制限する

次のように **autofill** プロパティを使用して、連続する数値または日付を作成できます。

```
With Worksheets("Sheet1").Range("A1")
        .Value = 1
        .AutoFill Destination:=.Resize(20, 1), Type:=xlFillSeries
End With
```

この変換は簡単です。**column size** パラメータは、変更がないため省略できます。

```
tell application "Microsoft Excel"
        tell range "A1" of worksheet "Sheet1"
                set value to 1
                autofill destination (get resize row size 20) type ¬
                        fill series
        end tell
end tell
```

入力規則を使用する

Validation オブジェクトを使用して入力規則を設定し、ユーザーのセルへの入力を制限することができます。たとえば、次のようにして入力を正の整数に制限できます。

```
With ActiveSheet.Range("A2").Validation
        .Delete  'remove current validation object
        .Add _
                Type:=xlValidateWholeNumber, _
                AlertStyle:=xlValidAlertStop, _
                Operator:=xlGreater, _
                Formula1:="0"
        .IgnoreBlank = True
        .InputTitle = ""
        .ErrorTitle = "Invalid Entry!"
        .InputMessage = "Enter an integer > 0"
        .ErrorMessage = "The entry MUST be a positive integer."
        .ShowInput = True
        .ShowError = True
End With
```

AppleScript ではプロパティを削除できません。削除できるのは要素だけです。**range** の構造では **validation** は読み取り専用のプロパティですが、**modify** コマンドを使用して必要な場合に変更を加えることができます。このコマンドのパラメータでは、**add data validation** で使用するパラメータとすべて同じものを使用します。この指定した部分以外では、スクリプトはマクロと同じです。

```
tell application "Microsoft Excel"
        tell validation of range "A2" of active sheet
                --can't delete , modify if need be
                add data validation type validate whole number ¬
                        alert style valid alert stop ¬
                        operator operator greater ¬
                        formula1 "0"
                set ignore blank to true
                set input title to ""
                set error title to "Invalid Entry!"
                set input message to "Enter an integer > 0"
                set error message to "The entry MUST be a positive integer."
                set show input to true
                set show error to true
        end tell
end tell
```

数値と文字列の書式を設定する

コードを使用して、簡単な方法でセルの書式を設定できます。

数値の表示形式

NumberFormat プロパティに、有効な書式の文字列を指定すると、GUI と同じようにセルの **number format** が設定されます。数値の表示形式は、セミコロンで 4 つの部分に区切られて表されます。既定では、1 番目は正の数、2 番目は負の数、3 番目は 0、4 番目は文字列に適用されます。文字列の書式を指定するには、@ 文字列を使用します。Visual Basic for Applications (VBA) では、次のようになります。

```
ActiveSheet.Range("A1").NumberFormat = "+0;-0;0;@"
```

AppleScript では次のようになります。

```
set number format of range "A1" of active sheet to "+0;-0;0;@"
```

また、セクションごとに適用を変えて、8 色あるユーザー定義のフォントの色のいずれかを指定することもできます。たとえば、-100 未満の数値をかっこで囲んだ赤色、-100 から +100 の値を緑色、100 を超える数値を赤色で表示するように設定できます。かっこのない表示形式には、かっこと同じ幅のスペースを右側に追加して、数値の配置を揃えます。VBA では次のようになります。

```
ActiveSheet.Range("A1:A100").NumberFormat = _
"[Red][<-100](0);[Green][<=100]0_);[Red]0_);@_)"
```

AppleScript では次のようになります。

```
set number format of range "A1:A100" of active sheet to ¬
        "[Red][<-100](0);[Green][<=100]0_);[Red]0_);@_)"
```

文字列の配置と折り返し

配置 (水平方向の配置の左詰め、中央揃え、右詰めなど) は、AppleScript の **horizontal alignment** プロパティと **vertical alignment** プロパティを使用して設定します。文字列の折り返しは、**wrap text** プロパティを使用して設定します。VBA では次のようになります。

```
With ActiveSheet.Range("A1:J10")
        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlCenter
        .WrapText = True
End with
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        tell range "A1:J10" of active sheet
                set horizontal alignment to horizontal align center
                set vertical alignment to vertical alignment center
                set wrap text to true
        end tell
end tell
```

horizontal alignment パラメータと **vertical alignment** パラメータの列挙はそれぞれ、*horizontal align center* と *vertical alignment center* になり、少し異なるので注意が必要です。

フォントの書式設定

フォントの書式設定は、セルの **Font** オブジェクトを使用して設定します。複数のプロパティを一度に適用できます。VBA では次のようになります。

```
With ActiveSheet.Range("A1").Font
        .Name = "Verdana"
        .FontStyle = "Bold"
        .Size = 12
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleDouble
        .ColorIndex = xlAutomatic
End With
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        tell font object of range "A1" of active sheet
                set {name, font style, font size, strikethrough, ¬
                        superscript, subscript, outline font, ¬
                        shadow, underline, font color index} to ¬
                        {"Verdana", "Bold", 12, false, false, ¬
                        false, false, false, underline style double, ¬
                        color index automatic}
        end tell
end tell
```

このコードでは、すべてのプロパティをリストとして一度に設定します。1 行ずつ指定した場合は、どのプロパティがどの値に設定されるかをより簡単に確認できます。

color index プロパティは、カラーパレットで使用できる 56 色を参照します。これらの色は、書式パレットやツールバーに表示される 16 色少ないパレット (40 色) ではなく、[書式] メニューのパレットで使用できる 56 色と同じです。

ただし、色の順番はまったく異なります。VBA と AppleScript に共通する "既定のパレット" が存在したのは何年も前のことです。インデックス番号付きのカラーパレットの正しいレイアウトについては、Excel 2004 AppleScript リファレンス (http://download.microsoft.com/download/3/A/7/3A70FB4B-0C6A-43E3-AAB7-AC9166B25632/Excel2004AppleScriptRef.pdf、英語) の「Text Suite Classes」 セクションにある「Class: font」で説明しています。

number format プロパティでは 8 色のユーザー設定色を設定できますが、**font color index** プロパティを使用してフォントの色を設定することもできます。VBA では次のようになります。

```
ActiveSheet.Range("A1").Font.ColorIndex = 3 'Default palette red
```

AppleScript では次のようになります。

```
set font color index of font object of range "A1" of active sheet to 3
```

font クラス (Text Suite) の **font color index** プロパティについて、カラー インデックスの列挙型を確認してください。*color index automatic* (通常は黒)、*color index none*、*a color index integer* の 3 つの定数があり、エラーが発生することなくコンパイルして実行できます。ただし、既存の色はすべて黒に変わるので、*color index automatic* が既定になります。

56 個の定数 (番号 `1` から `56`) では次のようになります。VBA では、オブジェクト ブラウザで確認すると、*xlAutomatic* と *xlNone* のみが表示されます。`xl` と定数 `vb` のすべての定数と同様に、これらは数値 (大きな負の数) で表すこともできます。ただし、番号 1 から 56 も予約されています。

AppleScript では、*a color index integer* は番号 1 から 56 のプレースホルダ定数です。これは単なる説明であって、実際の定数ではありません。

Microsoft の RGB 値を使用して色を設定できますが、Microsoft Excel ではパレットにある最も近い色が選択されます。まったく同じ色にはなりませんが、純色の赤、緑、青は、正確な色になります。VBA では次のようになります。

```
ActiveSheet.Range("A1").Font.Color = RGB(255, 0, 0)
```

AppleScript では次のようになります。

```
set color of font object of range "A1" of active sheet to {255, 0, 0}
```

色と罫線の書式を設定する

Visual Basic for Applications (VBA) でカラーパレットの色を変更するには、以下の手順を実行します (適用されるのは **workbook** のみです)。

```
ActiveWorkbook.Colors(33) = RGB(6, 8, 242)
ActiveSheet.Range("A1").Font.ColorIndex = 33 'Assign the new color
```

パレット上の特定スポットの色 (33) を、近似の RGB 色 {6 ,8, 242} に設定すると、以降その番号 33 **color index** を使用して、テキストや塗りつぶしにその色を設定できるようになります。この方法は **font** や他のオブジェクトに色を直接設定するよりも効率的です。

ただし、VBA の **Colors** プロパティは AppleScript で使用することができないため、別の方法またはこれに近い方法を見つける必要があります。AppleScript には、**Colors Collection** オブジェクトに相当するものがありません。また、色を作成したり削除したりできないため、色を要素にすることができません。色の数は 56 に固定されています。

ここで使用できるのは、**reset colors** コマンドの呼び出しです。これはカラーパレットを既定値に復元するコマンドです。**color** プロパティを色 (RGB) で設定することもできますが、ほとんどの場合は 56 色の中から 1 つを選択します。したがって、**color index** プロパティを使用してもかまいません。こうすると、取得する色をさしあたって正確に把握することができます。確実に目的の色を取得するためには、最初に **reset colors** コマンドを呼び出します。

色をブレンドすることもできます。以下の例では、セル内部、グラフ オブジェクト、描画オブジェクトへのグラデーションによる塗りつぶしのように、パターンやグラデーションが設定された部分で 2 色をブレンドしています。

背景の書式設定

セルの **interior object** プロパティを使用して、背景の色とパターンを設定します。

```
With ActiveSheet.Range("A1").Interior
        .ColorIndex = 3
        .Pattern = xlGray50
        .PatternColorIndex = 13
End With
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        tell interior object of range "A1" of active sheet
                set color index to 3
                set pattern to pattern gray 50
                set pattern color index to 13
        end tell
end tell
```

このコードにより、基本の内部色に、独自の色を使用した細かいパターンを必要に応じて重ね合わせて、さまざまなブレンド色を作成できます。

罫線

AppleScript では、すべての **border** を設定できますが、一度にすべてを設定する必要がある場合は、VBA で設定する場合よりもコードが長くなります。

VBA では、罫線はオブジェクト モデルの構成要素の 1 つであり、セルの **Borders Collection** と **Border** オブジェクトを使用して設定します。コレクション プロパティを使用すると、外枠の罫線を一度に設定できます。

```
With ActiveSheet.Range("D10").Borders
        .LineStyle = xlDouble
        .ColorIndex = 3
End With
```

各罫線を個別に設定することもできます。これは AppleScript での方法に近くなります。

罫線と対象オブジェクトの間には多対一のリレーションシップがあるように見えますが、罫線を **range** または他のオブジェクトの要素として実装することはできません。新しい罫線を作成したり既存の罫線を削除できないという点で、罫線は "読み取り専用" であるため、要素にはなりません。

罫線のプロパティは変更できるので、この点では、罫線は "書き込み可能" です。1 つの要素として定義されたものであれば、`make new border at range` を実行できそうですが、実際、新しい罫線を作成したり、削除したりすることはできません。罫線は常に存在します。

1 つの **border** プロパティに定数を列挙して、異なる種類の罫線を表すこともできません。罫線は、それぞれのプロパティを変更できる独立したオブジェクトの集まり、つまりコレクション型のオブジェクトである必要があります。AppleScript にこのようなオブジェクトはありません。また、AppleScript で、読み取り専用の組み込み要素を、このモデルに当てはめることはできません。これらの要素は、VBA ではコレクション オブジェクトとして扱われますが、AppleScript では、これらは **base object** 以外の親には属しません。同じ問題が Microsoft Word のヘッダーとフッターにも存在します。

どちらの問題に対しても、同じ方法で対処します。ただし、ヘッダーとフッターの場合とは異なり、罫線は 6 本ではなく 12 本あります。したがって、**get border** コマンドを使用して **which border** を 12 回指定し、各罫線を別々に設定する必要があります。Word でも、同様の方法で罫線が管理されます。

上に引用した VBA コードのように、**cell** と **range** のすべての罫線の **line style** と **color index** を設定するには、次のようにします。

```
tell application "Microsoft Excel"
        tell range "D10" of active sheet
                set allBorders to {} -- initialize a list
                set end of allBorders to get border ¬
                        which border border bottom
                set end of allBorders to get border ¬
                        which border border left
                set end of allBorders to get border ¬
                        which border border right
                set end of allBorders to get border ¬
                        which border border top
                set end of allBorders to get border ¬
                        which border diagonal down
                set end of allBorders to get border ¬
                        which border diagonal up
                set end of allBorders to get border ¬
                        which border edgebottom
                set end of allBorders to get border ¬
                        which border edge left
                set end of allBorders to get border ¬
                        which border edge right
                set end of allBorders to get border ¬
                        which border edge top
                set end of allBorders to get border ¬
                        which border inside horizontal
                set end of allBorders to get border ¬
                        which border inside vertical
        end tell
        repeat with thisBorder in allBorders
                try
                        set line style of thisBorder to double
                end try
                try
                        set color index of thisBorder to 3
                end try
        end repeat
end tell
```

ここでは、新しい罫線のそれぞれに対してリストの終了を設定しています。これは、リストを連結するよりも効率的な方法です。処理する項目が数個であれば目に見える違いはありませんが、このプロシージャ全体を数百回も繰り返すとなると、スクリプトの処理速度に大きな違いが生じることがあります。

line style は、他のプロパティと同様に、リスト全体に対して一度に設定することはできません。存在していない罫線を除外できても、スクリプトでエラーが発生します。このため、罫線ごとに `try/end try` を使用して、リストの各メンバに対して `repeat` ループを実行する必要があります。既定の標準シートのセルでは、両方のコマンドが最後の 2 本の罫線 (*inside horizontal* と *inside vertical*) でエラーが返されます。それぞれ **get border** コマンドを実行してオブジェクトが返されることがあっても、実際にはこれらの罫線が存在していない可能性があります。

リスト全体に対して一度に **set line style** などのコマンドを実行できないため、リストを使用して大きな利点があるわけではありません。代わりに、罫線を作成するたびに線のスタイルとカラー インデックスを設定 (**set**) し、ハンドラに各罫線を渡して処理することもできます。次に例を示します。

```
tell application "Microsoft Excel"
        tell range "D10" of active sheet
                set thisBorder to get border ¬
                        which border border bottom
                my ModifyBorder(thisBorder)
                set thisBorder to get border ¬
                        which border border left
                my ModifyBorder(thisBorder)
                set thisBorder to get border ¬
                        which border border right
                my ModifyBorder(thisBorder)
                set thisBorder to get border ¬
                        which border border top
                my ModifyBorder(thisBorder)
                set thisBorder to get border ¬
                        which border diagonaldown
                my ModifyBorder(thisBorder)
                set thisBorder to get border ¬
                        which border diagonal up
                my ModifyBorder(thisBorder)
                set thisBorder to get border ¬
                        which border edge bottom
                my ModifyBorder(thisBorder)
                set thisBorder to get border ¬
                        which border edge left
                my ModifyBorder(thisBorder)
                set thisBorder to get border ¬
                        which border edge right
                my ModifyBorder(thisBorder)
                set thisBorder to get border ¬
                        which border edge top
                my ModifyBorder(thisBorder)
                set thisBorder to get border ¬
                        which border inside horizontal
                my ModifyBorder(thisBorder)
                set thisBorder to get border ¬
                        which border inside vertical
                my ModifyBorder(thisBorder)
        end tell
end tell
```

```
on ModifyBorder(thisBorder)
        tell application "Microsoft Excel"
                try
                        set line style of thisBorder to double
                end try
                try
                        set color index of thisBorder to 3
                end try
        end tell
        return
end ModifyBorder
```

この方がより効率的ですが、どちらの方法も正しく機能します。ModifyBorder ハンドラまたは、または最初の方法では、repeat ループの中のコマンドを変更するだけで、すべての罫線に異なる効果を適用できます。

ほとんどの場合、プロパティを設定する罫線は 1 本または数本です。これを VBA で記述すると次のようになります。

```
With ActiveSheet.Range("D10")
        .Borders.LineStyle = xlNone ' Remove existing borders
        With .Borders(xlEdgeLeft)
                .LineStyle = xlContinuous
                .Weight = xlThin
                .ColorIndex = 3
        End With
        With .Borders(xlEdgeRight)
                .LineStyle = xlContinuous
                .Weight = xlThick
                .ColorIndex = 3
        End With
End With
```

AppleScript で既存のすべての罫線を削除する場合、上に示したいずれかのシーケンスを実行しますが、repeat ループまたは ModifyBorder ハンドラ内のコマンドを次のように変更します。

```
set line style of thisBorder to line style none
```

続けて以下のように記述します。

```
tell application "Microsoft Excel"
        tell range "D10" of active sheet
        set thisBorder to get border ¬
                which border edge left
                tell thisBorder
                        set line style to continuous
                        set weight to border weight thick
                        set color index to 3
                end tell
                set thisBorder to get border ¬
                        which border edge right
                tell thisBorder
```

```
                        set line style to continuous
                        set weight to border weight thick
                        set color index to 3
                end tell
        end tell
end tell
```

2 本の外枠罫線のそれぞれに対して同じ 3 つの設定を適用しているので、ここでもハンドラを使用してこの操作をカスタマイズすると効率的です。

```
tell application "Microsoft Excel"
        tell range "D10" of active sheet
                set thisBorder to get border ¬
                        which border edge left
                my Customize(thisBorder)
                set thisBorder to get border ¬
                        which border edge right
                my Customize(thisBorder)
        end tell
end tell

on Customize(thisBorder)
        tell application "Microsoft Excel"
                tell thisBorder
                        set line style to continuous
                        set weight to border weight thick
                        set color index to 3
                end tell
        end tell
end Customize
```

Microsoft Excel アプリケーション ブロックの外へも、アプリケーションのオブジェクトをパラメータとして外部ハンドラに渡すことができます。ハンドラは、呼び出されるとその項目を `border id border edge left of cell "D10" of active sheet of application "Microsoft Excel"` として認識し、ハンドラ内のアプリケーションの `tell` ブロックに戻すことができます。

条件付き書式

条件付き書式は、Excel Suite の **format condition** オブジェクトを使用して適用されます。このオブジェクトは、**range** クラスの要素です。クラスには、0 または複数のオブジェクトを含めることができます (ここでは、1 つの range クラスに対してオブジェクトは 3 つに限定されています)。条件付き書式ごとに最大 3 つの条件を設定できます。設定できる書式は、フォント、内部 (セルの背景色)、および罫線です。たとえば、Visual Basic for Applications (VBA) の次のコードでは、緑の太字のフォントを 0 ～ 10 の値に適用します。

```
With ActiveSheet.Range("B1").FormatConditions
        .Delete 'Delete existing conditional formats
        With .Add(Type:=xlCellValue, _
                Operator:=xlBetween, _
```

```
                Formula1:="0", _
                Formula2:="10").Font
        .Bold = True
        .Italic = False
        .ColorIndex = 4
        End With
End With
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        tell range "B1" of active sheet
                try
                        delete (every format condition)
                end try
                set newFormatCondition to make new format condition ¬
                        at end with properties {format condition type: ¬
                        cell value, condition operator:operator between, ¬
                        formula1:"0", formula2:"10"}
                tell font object of newFormatCondition
                        set {bold, italic, font color index} ¬
                                to {true, false, 4}
                end tell
        end tell
end tell
```

書式の条件は **range** オブジェクトの要素であり、プロパティではないため、新しい条件を作成する前に、既にある条件を削除できます。これは、他の人から渡されたり、何度も使用したブックで、初めて範囲に書式条件を作成する場合に便利です (条件が設定された範囲に別の書式条件を作成する場合、設定済みの条件書式を保持するには、**delete** コマンドを省略します)。**delete** コマンドは `try` ブロックに配置します。これで、既存の書式条件がない場合はエラーが返されます。

Dictionary によると、**format condition** 要素のすべてのプロパティは読み取り専用になっているため、設定することはできません。しかし、他のアプリケーションでは、Dictionary に読み取り専用として一覧表示されていても、`make new with properties {...}` を使用して開始時にのみ設定できるプロパティを持つクラスが数多くあります。

このようなクラスは、Microsoft Excel、Word、PowerPoint ではほとんど見られませんが、Microsoft Entourage など、"より標準的" なスクリプト対応アプリケーションには多数存在します。これらについては、常に確認およびテストを行うよう心がけてください。

format condition type、**condition operator**、**formula1**、**formula2** は、ここで最初に設定できるプロパティであり、列挙型のテキスト プロパティです。つまり、これらが条件になります。ここで、`make new format condition` のプロパティ内に **font object** または **interior object** の設定を含めても無視されます。フォント オブジェクト自体は設定 (**set**) できませんが、代わりにフォント オブジェクトを含む **format condition** 要素を作成すると、個別のプロパティ (**bold**、**italic**、**name**、**font color index** など) を設定できるようになります。この方法は、Microsoft Office アプリケーションでは一般的です。フォントなどの複雑なオブジェクトはすべて、他のオブジェクトでプロパティを保持できます。

オブジェクトではなく、後で設定できる個別のプロパティを持たない "単純な" な読み取り専用のプロパティも、最初に設定できます。このことにより、**format condition** オブジェクトを利用してスクリプトを作成できるようになっています。

これらの要素は、範囲の `at beginning` または `at end` に作成する必要があります。競合する条件が 2 つ重なる場合を除き、どちらに作成するかは問題ではありません。

また、Dictionary では `formula 1` と `formula 2` はパラメータ名として記載されていますが、これらは **add data validation** に表示されるように、スペースなしで **formula1** と **formula2** にコンパイルされます。つまり、これらは同義語として実装されているため、どちらのバージョンを入力しても問題ありません。

数式を表す *expression* (VBA では *xlExpression*) で **Type** を宣言することにより、数式条件に基づいてさらに複雑な書式を適用できます。

```
With ActiveSheet.Range("B1").FormatConditions
        .Delete
        With .Add(Type:=xlExpression, _
                Formula1:="=AND($A1>0,$A1<10)")
                With .Font
                        .Bold = True
                        .Italic = False
                        .ColorIndex = 2
                End With
                .Interior.ColorIndex = 3
        End With
End With
tell application "Microsoft Excel"
        tell range "B1" of active sheet
                try
                        delete (every format condition)
                end try
                set newFormatCondition to make new format condition ¬
                        at end with properties {format condition type:¬
                        expression, formula1: "=AND($A1>0,$A1<10)"}
                tell newFormatCondition
                        tell its font object
                                set {bold, italic, font color index} ¬
                                        to {true, false, 2}
                        end tell
                        set color index of its interior object to 3
                end tell
        end tell
end tell
```

このコードを実行すると、`A1` に 1 ～ 9 の数値がある場合、`B2` でセルの背景色が赤色になり、フォントが太字の白になります。AppleScript のコードも、問題なく機能します。

書式条件では、罫線、フォント、内部 (セルの背景色) の書式を設定できます。VBA では、**Borders Collection** は **FormatCondition** オブジェクトのプロパティで、フォントや内部と同じように機能します。AppleScript では、Microsoft Excel Suite にあるように、**format condition** 要素を対象にした **get border** コマンドを使用する必要があります。別のスイートでは別の **get border** コマンドとして記載されていますが、これは、対象オブジェクトの種類 (**range** ではなく **format condition**) を指定できるようにするためです。ただし、実際には従来の **format condition** コマンドと同じで、範囲で使用したものと同じように機能します。このコマンドは、**which border** パラメータで指定した種類の罫線を返すので、ここから取得した罫線ごとに書式を適用します。

グラフを作成する

グラフは柔軟性が高いこともあり、オブジェクトとしては複雑です。**chart object** オブジェクトは **chart sheet** として、またはワークシートに埋め込まれた **chart object** として存在します。グラフ オブジェクトには **chart** プロパティがあります。このプロパティは数多くのプロパティを持つ **chart** (クラス) です。

次の例では、作業中のワークシートに *XY Scatter* **chart** を埋め込みます。ここでは、データが列に含まれ、最初の **row** がヘッダーであることを想定しています。ヘッダーの数から **series** の数が自動的に決まり、`column A` の **X** 値の数からデータの個数が決まります。

このトピックの最後に、この例のデータを構成する数値の表を示します。オンラインでこのトピックを参照している場合は、データをコピーして Microsoft Excel ワークシートにペーストし、スクリプトを実行できます。数値の正確さはあまり重要ではありません。`columns A、B、C` には `rows 2 〜 41` に 1 〜 40 が入力されています。`Column D` でも、`row 2 〜 41` に数式 `=MAX(A:A)/2+(1+SIN(A2)*2*PI())` が入力されています。`Column E` では、同じ 1 〜 40 の数値がランダムに分布しています (このため、`Y4` で作成されるグラフは数値の変化に富んだものになります)。

```
Public Sub CreateEmbeddedChart()
        Dim oChartObj As ChartObject
        Dim rHeaders As Range
        Dim rData As Range
        Dim i As Long

        ' Dynamically grab the data in columns
        With ActiveSheet
        'Find the headers in the first row
        Set rHeaders = .Range("A1").Resize(1, _
                 .Cells(1, .Columns.Count).End(xlToLeft).Column)
        'Now find the number of data rows.
        Set rData = .Range("A2:A" & .Range("A" & _
                 .Rows.Count).End(xlUp).Row).Resize(, rHeaders.Count)
        If rHeaders.Columns.Count = 1 Or _
                 rData.Rows.Count = 1 Then Exit Sub  'no data

        'Create chart object
        Set oChartObj = .ChartObjects.Add( _
                 Left:=400, _
                 Top:=100, _
```

```
            Width:=500, _
            Height:=400)

    'Now build the Chart within the ChartObject
    With oChartObj.Chart
            .ChartType = xlXYScatterSmooth 'define chart type

            'Add each series
            For i = 2 To rHeaders.Count
                    With .SeriesCollection.NewSeries
                            .Values = rData.Columns(i)
                            .XValues = rData.Columns(1)
                            .Name = rHeaders.Cells(i)
                    End With
            Next i

            'Add Titles and Format Chart and Axes
            .HasTitle = True
            With .ChartTitle
                    .Caption = "My XY Scatter Chart"
                    .HorizontalAlignment = xlHAlignCenter
                    With .Font
                            .Name = "Verdana"
                            .Size = 12
                            .Bold = True
                    End With
            End With
            With .Axes(xlCategory, xlPrimary)
                    .HasTitle = True
                    With .AxisTitle
                            .Caption = "X Values"
                            With .Font
                                    .Name = "Arial"
                                    .Size = 10
                                    .Bold = True
                            End With
                    End With
            End With
            With .Axes(xlValue, xlPrimary)
                    .HasTitle = True
                    With .AxisTitle
                            .Caption = "Y Values"
                            With .Font
                                    .Name = "Arial"
                                    .Size = 10
                                    .Bold = True
                            End With
                    End With
            End With

            'Format Plot area
            With .PlotArea
```

```
                With .Border
                        .Color = vbBlue
                        .LineWeight = 1
                        .LineStyle = xlContinuous
                        .Transparency = 0
                End With
                With .Fill
                        .Visible = True
                        .ForeColor.RGB = RGB(150, 200, 255)
                        .Transparency = 0.5
                End With
        End With

        'Format Legend
        .HasLegend = True
        With .Legend
                .Position = xlRight
                With .Border
                        .Color = vbBlue
                        .LineWeight = 1
                        .LineStyle = xlContinuous
                        .Transparency = 0
                End With
                With .Fill
                        .Visible = True
                        .ForeColor.RGB = RGB(150, 200, 255)
                        .Transparency = 0.5
                End With
                With .Font
                        .Name = "Arial"
                        .Size = 10
                        .ColorIndex = 5
                End With
        End With
    End With
    End With
End Sub
```

AppleScript では次のようになります (両方を試す場合は別のシートを使用してください)。

```
tell application "Microsoft Excel"
    --dynamically grab the data in columns
    tell active sheet
    --Find the headers in the first row:resize range A1
    --to last used cell of row 1
        set rHeaders to get resize range "A1" column size ¬
            (first column index of (get end (cell ¬
            (count columns) of row 1) ¬
            direction toward the left))
        --Now find the number of data rows
        --and set rData to whole range from row 2
        set rData to get resize range ("A2:A" & first row index ¬
            of (get end range ("A" & (count rows)) direction ¬
```

```
        toward the top)) column size (count (columns ¬
        of rHeaders))
if (count (columns of rHeaders)) = 1 or (count ¬
        (rows of rData)) = 1 then return --  no data, so quit

--Create chart object
set oChartObj to make new chart object at end ¬
        with properties {left position:400, top:100, ¬
        width:500, height:400}

--Now build the Chart within the ChartObject
tell chart of oChartObj
        set chart type to xy scatter smooth -- define chart type

        -- add each series
        repeat with i from 2 to (count (columns of rHeaders))
                set newSeries to make new series at end ¬
                        with properties {series values:¬
                        (column i of rData), xvalues:¬
                        (column 1 of rData), name:¬
                        (value of cell i of rHeaders)}
                --  need to specify _value_ (no default property)
        end repeat

        --'Add Titles and Format Chart and Axes
        set has title to true
        tell its chart title -- needs 'its !!
                set caption to "My XY Scatter Chart"
                tell font object
                        set name to "Verdana"
                        set font size to 12
                        set bold to true
                end tell
        end tell

        set categoryAxis to get axis axis type category axis ¬
                which axis primary axis
        tell categoryAxis
                set has title to true
                tell its axis title -- needs 'its' !!
                        set caption to "X Values"
                        tell font object
                                set name to "Arial"
                                set font size to 10
                                set bold to true
                        end tell
                end tell
        end tell

        set valueAxis to get axis axis type value axis ¬
                which axis primary axis
        tell valueAxis
```

```
                set has title to true
                tell its axis title -- needs 'its' !!
                        set caption to "Y Values"
                        tell font object
                                set name to "Arial"
                                set font size to 10
                                set bold to true
                        end tell
                end tell
        end tell
        --'Format Plot area
        tell plot area object
                tell its border -- needs its
                        set color to {0, 0, 255}
                        -- will this work?, if not:
                        --set its color index to 5
                        set line weight to 1
                        set its line style to continuous
                        -- no transparency property in AppleScript
                        --set transparency to 0
                end tell
                tell its chart fill format object
                        set visible to true
                        --set fore color to {150, 200, 255}
                        --can't set color
                        --set transparency to 0.5
                        set foreground scheme color to 23
                        set transparency to 0.8 -- the same color
                end tell
        end tell

        --format legend
        set has legend to true
        tell legend object
                set its position to legend position right
                tell its border -- needs its
                        set color to {0, 0, 255}
                        -- will this work?, if not:
                        --set its color index to 5
                        set line weight to 1
                        set its line style to continuous
                        -- no transparency property in AppleScript
                        --set transparency to 0
                end tell
                tell its chart fill format object
                        set visible to true
                        --set fore color to {150, 200, 255}
                        --can't
                        --set transparency to 0.5
                        set foreground scheme color to 23
                        set transparency to 0.8 -- the same color
                end tell
```

```
                                    tell its font object
                                            set name to "Arial"
                                            set font size to 10
                                            set its font color index to 5
                                    end tell
                            end tell
                    end tell
            end tell
    end tell
```

Visual Basic for Applications (VBA) マクロではできて、このスクリプトではできないことが 2 つあります。1 つ目は、AppleScript には **border** の `transparency` プロパティがなく、枠線の透明度を設定できないということです。実行してみると、マクロでは透明度が 0 (不透明) に設定されますが、これは既定値であるため、変化はわかりません。

2 つ目は、**chart fill format** クラスの **fore color** プロパティに関することです。このプロパティは、グラフの **plot area** (グラフの塗りつぶしの色) と **legend** (グラフの右側にある凡例ボックス) の両方の **chart fill format object** プロパティのクラスですが、AppleScript では読み取り専用になっています。これを直接設定することはできず、代わりに設定できる **color index** プロパティもありません。

ただし、別の方法として、**foreground scheme color** インデックスを設定できます。インデックス付きのカラーの表はなく、この場合、前景の配色 `23` と透明度の設定 `0.8` は、マクロで作成された色と網かけ `RGB(150, 200, 255)` に一致します。

配色のインデックスは 100 以上あり、Excel のどの色も試行錯誤して対応色を探すことができます。また、この他にグラフの **chart group** プロパティに設定できる 17 の配色があり、これにより無数の色が存在します。

前景色を設定する行は、コンパイルされますが、削除する必要があります。削除しないと、スクリプトでエラーが発生します。コード サンプルでは、2 回ともコメントアウトされ、その代わり前景の配色が設定されています。ここで見られる AppleScript と VBA の違いは、AppleScript では、この **foreground scheme color** プロパティが単純な色の型 (3 つの整数から成る {r, g, b} のリスト) であるのに対し、VBA では、色を設定するためにさらに **RGB** プロパティを必要とする **ColorFormat** オブジェクトがあるという点です。

スクリプトには、マクロ構文を模倣するとエラーが発生する箇所が 2 か所あります。1 つは最初の方にある次の行です。

```
Set rData = .Range("A2:A" & .Range("A" & _
        .Rows.Count).End(xlUp).Row).Resize(, rHeaders.Count)
```

もう 1 つは、後続のブロックの最後の行です。その下の数行で系列を追加しています。

```
With .SeriesCollection.NewSeries
        .Values = rData.Columns(i)
        .XValues = rData.Columns(1)
        .Name = rHeaders.Cells(i)
End With
```

これらを直接次のように変換するとします。

```
set rData to get resize range ("A2:A" & first row index of (get end ¬
        range ("A" & (count rows)) direction toward the top)) ¬
        column size (count rHeaders)
```

および

```
set newSeries to make new series at end with properties ¬
        {series values:(column i of rData), xvalues:¬
        (column 1 of rData), name:(cell i of rHeaders)}
```

1 つ目の例では致命的なエラーが発生します。2 つ目の例では [凡例] ボックスの系列の名前 (`Y1, Y2, Y3, Y4`) が取得されず、代わりに既定の名前 `Series1, Series2, Series3, Series4` が表示されます。

これは、VBA では多くのオブジェクトに既定のプロパティが存在し、これらについてはコードに記述する必要がないためです。行、または **range** の既定のプロパティは、**Cells** プロパティです。`rHeaders` の実体は行方向のみの範囲であるため、範囲をカウント (**Count**) することはセルをカウントすることになり、`rHeaders.Cells.Count` と同じ結果になります。

しかし、AppleScript には既定のプロパティが存在せず、範囲をカウントする意味がないため、スクリプト エラーとなります。スクリプトに示されているように、これは `count cells of rHeader` または `count columns of rHeader` に変更する必要があります。

新しい系列を作成する場合、凡例の名前が失われます。これは、セルの既定のプロパティはそのセルの **Value** プロパティでなければならないためです。したがって、`.Name = rHeaders.Cells(i)` (**Name** は **String** 型) を設定すると、系列の名前は確かに文字列 `Value of Cell(i)` に設定されます。

しかし、セルの既定のプロパティがない AppleScript では、名前を文字列や Unicode テキストではなく、セル (範囲) に設定しようとした場合、そのとおりにはなりません。エラーは発生しませんが、何も起こりません (後で名前を求められた場合にはエラーが発生する可能性があります)。名前は、`Series1,...` などの既定値のままです。`newSeries` の名前を `value of cell i of rHeaders` に設定すると、うまく行きます。

重要

ワークシートで新しいグラフ オブジェクトを作成したり、グラフで新しい系列を作成したりする際は、どちらの場合も必ず `at end` (または `at beginning`) を指定する必要があります (これらが `tell` ブロックで指定されていない場合は、それぞれ `at end of active sheet` と `at end of oChartObj` が使用されます)。これら両方の要素のプロパティは、`make new chart object with properties` ステートメントおよび `make new series with properties` ステートメントで、オブジェクトの作成と同時に設定できます。

最後の注意点として、多くの異なるオブジェクトの数多くのプロパティには、プロパティ自体の用語と同じ '型' (クラス) の用語があるということがあります。たとえば、**chart title**、**axis title**、および **line style** などのプロパティは、すべて同じ名前のクラスを参照します。**font** クラス型の **font object** プロパティ、**legend** クラス型の **legend object** プロパティなどに比べるとこちらの方が扱いやすいように見えますが、これらの構造では、スクリプト作成者の混乱を軽減しています。

プロパティの名前とクラスの名前が同じである場合、AppleScript では 2 つの状況で混乱が生じる可能性があります。まず、`whose` 句内で常に混乱が生じます。また、`of` 構造ではなく、プロパティの親オブジェクトに対して `tell` ブロックを使用する場合に混乱が生じる場合があります。このスクリプト全体で見られるように、VBA スクリプト作成者が AppleScript を記述するときには、VBA の `With` ブロックを、ネストされた `tell` ブロックに書き直すことがよくあります。この場合は注意が必要で、プロパティの前に `its` を使用する必要があります。`its` は、使用しても問題は起こらないため、スクリプトにはいくつか追加されています。また、`tell` ブロック内で親オブジェクトに対して使用しても問題が起こることはありません。

グラフについて学ぶことはまだたくさんあります。思わぬ障害に遭遇した際は、クリエイティブに解決策を考え、構文で近道をしようとして方法を見失っていないかどうかを客観的に確認してください。

グラフの例をテストするための数値を以下に示します。

Xs	Y1	Y2	Y3	Y4
1	1	40	26.29	28
2	2	39	26.71	32
3	3	38	21.89	16
4	4	37	16.24	38
5	5	36	14.97	22
6	6	35	19.24	17
7	7	34	25.13	6
8	8	33	27.22	7
9	9	32	23.59	4
10	10	31	17.58	14
11	11	30	14.72	18
12	12	29	17.63	23
13	13	28	23.64	8
14	14	27	27.22	40

15	15	26	25.09	20
16	16	25	19.19	35
17	17	24	14.96	2
18	18	23	16.28	21
19	19	22	21.94	11
20	20	21	26.74	31
21	21	20	26.26	37
22	22	19	20.94	5
23	23	18	15.68	36
24	24	17	15.31	15
25	25	16	20.17	30
26	26	15	25.79	27
27	27	14	27.01	13
28	28	13	22.70	12
29	29	12	16.83	3
30	30	11	14.79	24
31	31	10	18.46	39
32	32	9	24.46	19
33	33	8	27.28	33
34	34	7	24.32	1
35	35	6	18.31	10
36	36	5	14.77	25

37	37	4	16.96	34
38	38	3	22.86	29
39	39	2	27.06	9
40	40	1	25.68	26

ページをプリントする

プリントとページ設定について

プリント

プリントは **print out** コマンドを使用して実行します。通常、これは **sheet** に次のように適用されます。

```
ActiveSheet.PrintOut
print out active sheet
```

print だけでは使用しないでください。 このコマンドは Standard Suite にある必須の **print** コマンドなのでプリントは実行されますが、このコマンドでは UI に標準の [プリント] ダイアログが表示され、ユーザーが [OK] をクリックするまで待機することになります。また、**print out** コマンドで使用できるさまざまな制御用のオプション パラメータも利用できません。

たとえば、オプションとして部数とページ範囲を指定できます。**to** パラメータを省略すると、UI での操作と同様に最後のページまでプリントします。

```
ActiveSheet.PrintOut From:=2, To:=2, Copies:=4
print out active sheet from 2 to 2 copies 4
```

シートがグループ化されている場合は、print out を **workbook** または **active window** に適用することで、すべてプリントすることができます。Visual Basic for Applications (VBA) では、これらは別々のメソッドとして指定します。

```
ActiveWorkbook.PrintOut
print out active workbook
```

既定では、ヘッダーまたはフッターにページ番号がある場合、シートに連続したページ番号が付いてプリントされます。この動作はオプションで変更できます。

AppleScript Dictionary で、*sheet* / *window* / *workbook* がダイレクト パラメータ **printout options** の定数として一覧表示されていますが、これは誤りです。プリントの対象は定数 *sheet* ではなく、実際の **sheet** オブジェクトなので、これは正確な記述ではありません。この一覧には型 (クラス) **sheet** が型 **window** および型 **workbook** と共に記載されている必要があり、列挙定数は正しくありません。Dictionary では、説明の中で、適用できるクラスが示されています。

AppleScript Dictionary では、これまでの慣習として、複数のクラスが一覧にされることはありませんでした。そのため古い Dictionary では、型が `reference` となっていることが多くあります。これは "何らかのアプリケーション オブジェクト" を意味します。または、`anything` となっていることもあります。これは、許可される型の 1 つが **string**、**number** などの基本の AppleScript 型である場合に該当します。"anything" は、汎用的な型として使用されています。これは、適切なクラスのオブジェクトであれば、コマンドのダイレクト オブジェクトになることができ、エラーは起こらないことを示すものです。これは "すべてのアプリケーション オブジェクト" ではなく、単に "「不特定」のアプリケーション クラスの複数の型" を意味しています。

Microsoft Excel と Microsoft Word の Dictionary では、別のスイートで、同じコマンドが異なる内容で記述されている場合もあります (**get border** など)。このため、Dictionary ではスイートごとに 1 つのクラスがダイレクト パラメータとして記載されていることがあります。これらは相互参照されており、正常に機能します。

Microsoft Office 2004 では、スクリプトによる PDF へのプリントは実行できません。**print out** コマンドの **print to file** パラメータで PDF へのプリントを実行できるように見えますが、実行すると紙にプリントされます。

ページ設定

VBA の **Page Setup** では、各パラメータで **Print Driver** を個別に呼び出す必要がありますが、GUI ダイアログで設定できるパラメータであればどのパラメータでも設定できます。

```
With ActiveWorkbook.Sheets("Sheet1").PageSetup
        .PrintTitleRows = "$1:$2"
        ActiveSheet.PageSetup.PrintArea = ""  'determine at print time
        .LeftHeader = ""
        .CenterHeader = "My Report"
        .RightHeader = ""
        .LeftFooter = "Page &P of &N"
        .CenterFooter = ""
        .RightFooter = "&Z&F"
        .LeftMargin = Application.InchesToPoints(1)
        .RightMargin = Application.InchesToPoints(1)
        .TopMargin = Application.InchesToPoints(1.5)
        .BottomMargin = Application.InchesToPoints(1.5)
        .HeaderMargin = Application.InchesToPoints(0.5)
        .FooterMargin = Application.InchesToPoints(0.5)
        .PrintHeadings = False
        .PrintGridlines = False
        .PrintComments = xlPrintNoComments
        .PrintQuality = -4
        .CenterHorizontally = False
        .CenterVertically = False
        .Orientation = xlLandscape
        .Draft = False
        .FirstPageNumber = xlAutomatic
        .Order = xlDownThenOver
        .BlackAndWhite = False
        .Zoom = 100
End With
```

ページを特定の高さまたは幅に合わせるには、**Zoom** プロパティを *False* に設定する必要があります。

```
With ActiveWorkbook.Sheets("Sheet1").PageSetup
        .Zoom = False
        .FitToPagesWide = 1
        .FitToPagesTall = 2
End With
```

これは AppleScript に簡単に書き直すことができます。

```
tell application "Microsoft Excel"
        tell page setup object of active sheet
                set print title rows to "$1:$2"
                set print area to "" -- determine at print time
                set left header to ""
                set center header to "My Report"
                set right header to ""
                set left footer to "Page &P of &N"
                set center footer to ""
                set right footer to "&Z&F"
                set left margin to inches to points inches 1.0
                set right margin to inches to points inches 1.0
                set top margin to inches to points inches 1.5
                set bottom margin to inches to points inches 1.5
                set header margin to inches to points inches 0.5
                set footer margin to inches to points inches 0.5
                set printheadings to false
                set print gridlines to false
                set print quality to -4
                set center horizontally to false
                set center vertically to false
                set page orientation to landscape
                set draft to false
                set first page number to 1
                set order to down then over
                set black and white to false
                set zoom to 100
        end tell
end tell
```

異なる点は、AppleScript では定数 *xlAutomatic* に相当する正式な用語がないことだけです。そのため、ページ番号は通常の "自動" である `1` から開始されます。*xlAutomatic* の VBA での定数 *Long* は -4105 です。これは AppleScript でも機能する可能性があります。

改ページ、ヘッダーとフッター、プレビュー

改ページ

page break プロパティを使用して、改ページを設定または解除できます。次のコードは、column C の左側と row 10 の上までを 1 ページとして改ページを設定します。

```
With ActiveWorkbook.Sheets("Sheet1")
        .Cells.PageBreaks - xlNone 'Clear all manual pagebreaks
        .Range("C10").PageBreak = xlPageBreakManual
End With
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        tell sheet "Sheet1" of active workbook
                set pagebreak of every range to page break none
                set page break of range "C10" to page break manual
        end tell
end tell
```

ここでは、every cell ではなく、page break of every range を使用します。

Visual Basic for Applications (VBA) では、ワークシートの **HPageBreaks** または **VPageBreaks** コレクションへの追加として、**horizontal page break** または **vertical page break** を指定することもできます。

```
With ActiveWorkbook.Sheets("Sheet1")
        .HPageBreaks.Add Before:=.Range("C10")
End With
```

AppleScript では、**horizontal page break** と **vertical page break** は別のクラスなのでより簡単です。

```
tell sheet "Sheet3" of active workbook
        make new horizontal page break at end with properties ¬
                {location:range ("C10")}
end tell
```

メモ at end または at beginning の挿入位置の指定は必須です。指定しない場合、スクリプト エラーが発生します。

また、その他のプロパティを指定することもできます。

```
tell application "Microsoft Excel"
        set page break of every range of active sheet to page break none
        make new horizontal page break at end of active sheet ¬
                with properties {location:range ("C10"), ¬
                        extent:page break full, horizontal page break type:¬
                        page break manual}
end tell
```

異なるヘッダーとフッターを設定する

複数ページをプリントするときに、先頭ページだけにヘッダーまたはフッターが必要な場合があります。Microsoft Excel では、Microsoft Word のように先頭ページや偶数ページのヘッダーとフッターを別に指定することができません。先頭ページにのみフッターをプリントするには、プリントの工程を 2 回に分けて、最初にフッターがある先頭ページをプリントします。次に、残りのページのフッターを削除してからプリントし、最後にシートにフッターを復元する必要があります。

```
Public Sub FirstPageFooterOnly_OneSheet()
        Dim sFooter As String
        With ActiveWorkbook.Sheets("My Sheet")
                  sFooter = .PageSetup.LeftFooter 'Save footer
                  .PrintOut From:=1, To:=1
                  .PageSetup.LeftFooter = "" 'Remove from remaining pages
                  .PrintOut From:=2
                  .PageSetup.LeftFooter = sFooter 'Restore footer
        End With
End Sub
```

次のコードは、AppleScript にそのまま変換したものです。Dictionary に類似の名前があり、正常に動作します。

```
tell application "Microsoft Excel"
        tell sheet "My Sheet" of active workbook
                  set sFooter to left footer of page setup object
                  --save footer
                  print out from 1 to 1
                  set left footer of page setup object to ""
                  -- remove footer 2nd page
                  print out from 2
                  set left footer of page setup object to sFooter
                  -- restore
        end tell
end tell
```

フッター (**left footer**、**center footer**、**right footer**) がある場合は、すべて保存して復元する必要があります。

プリント プレビュー

print out メソッドを使用して、Excel 内で **print preview** を作成できます。

```
ActiveSheet.Printout Preview:=True
print out active sheet with preview
```

これが操作を実行するコードですが、このスクリプトは (おそらくはプリンタ ドライバからの) 応答を待機しているような状態になり、ハングします。これを回避する方法を次に示します。これで、スクリプトは直接次の行に進むか、完了した場合は終了することができます。

```
tell application "Microsoft Excel"
        activate
        ignoring application responses
                print out active sheet with preview
        end ignoring
end tell
```

新しいバージョンの Microsoft Office で Excel にスクリプト メニューが導入され、そのメニューからスクリプトを実行するようになった場合は、`ignoring application responses` は無効となります。これは、スクリプトを実行しているアプリケーション以外のアプリケーションに対してのみ有効となります。このため、このスクリプトはシステムのスクリプト メニューから実行してください。

コメントとハイパーリンク

コメントはセルに添付されますが、実際は **cell** (**range**) ではなく、**sheet** の要素です。コメントの挿入には **add comment** コマンドを使用しますが、その方法は AppleScript と Visual Basic for Applications (VBA) とでは異なります。

```
Public Sub AddComment()
        Dim cmt As Comment
        With Worksheets("Sheet1").Range("A10")
        Set cmt = .Comment ' Check to see if comment exists
        If cmt Is Nothing Then _  ' if not, add one
                Set cmt = .AddComment
        End With
        With cmt
                .Text Text:="My Comment"  ' Overwrite existing comment
                .Visible = True
                .Shape.Select  'Select to allow editing
        End With
End Sub
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        tell range "A10" of active sheet
                set cmt to its Excel comment
                -- no error if nothing (dummy comment)
                set vis to visible of cmt
                --get any property , returns missing value if empty
                if vis is missing value then
                        set cmt to add comment
                end if
        end tell
```

```
        tell cmt
                Excel comment text text "My  comment"
                --overwrites, but omit 'over write'!
                set visible to true
                select its shape object -- to allow editing
        end tell
end tell
```

VBA では、コメントがあると仮定して変数を設定し、それが `Nothing` の場合、コメントを追加 (**Add**) します。通常、AppleScript で同様の処理を実行する場合は `try/error` ブロックを使用し、値がない場合はエラーが返されるものとして、`on error` ブロックにコメントを追加することを考えます。この場合、値がないかどうかを確認するため、場合によっては値を設定した変数を呼び出す必要があります。

しかし、**Excel comment** の動作は通常と異なり、セルにコメントがない場合でも、**Excel comment** プロパティを疑似的に使用できます。こちらではエラーは発生しません。ただし、範囲を対象とする `tell` ブロックの場合は `its` (`its Excel comment`) を含める必要があります。そうしないと、本当にコメントがある場合でもエラーが発生し、コメントを取得できません。

また、この疑似的な Excel コメントのすべてのプロパティは `missing value` を返します。このため、**visible** プロパティを確認し、*true* や *false* ではなく `missing value` が返される場合は、コメントを追加する必要があります。

コメントを追加するには、Excel Suite の **add comment** コマンドを使用します。**comment text** パラメータを指定してその場でテキストを追加することもできますが、この段階では本当にコメントを含むセルがまだ処理されていないため、これを行おうとするとスクリプトが複雑になります。このため、VBA マクロと同じようにコメントを追加し、`if/end if` ブロックが終了した後ですべてのコメントを同じ方法で処理します。

`if` ブロックが終了した段階で、このセルには疑似的なコメントではなく有効な Excel コメントが存在することが保証されます。最初から **visible** プロパティに適切な **value** (*true* または *false*) があったのか、(テキストを含まない) 新しいコメントを作成したかは関係なく、どちらも本当のコメントです。

ここで、**Excel comment text** コマンドを呼び出します。これは、コメントのプロパティを設定できる VBA コードとは異なります。**Excel comment** クラスには **author**、**shape object**、**visible** の各プロパティがありますが、テキスト用のプロパティがありません。

そこで、**Excel comment text** コマンドを代わりに使用します。パラメータを使用しない場合、このコマンドは既存のテキストを返します。そこで **text** パラメータでテキストを指定すると、古いテキストを返すのではなく、新しいテキストを書き込むようになります。また、ブール型のパラメータ **over write** (over と write の間にはスペース) もありますが、これを指定すると (*with* **over write**。**over write** *true* と同義)、この行で `range "A10" does not understand the Excel comment text message` というエラーが発生します。

この問題を解決するには、余分な **over write** パラメータを省略します。そうすると正常に機能します。**text** パラメータを指定して、**start** パラメータを含めない場合は、現在のテキストが上書きされます。既存のテキスト全体を置き換える場合は **start** パラメータは必要ありません。*start 1* が既定です。

コメントを編集する

VBA では、テキストを追加する以外にも、コードを使用してコメントを書式設定できます。しかし、AppleScript では **text frame** の文字にアクセスできないため、フォントを変更することはできません。`characters of text frame of shape object of theComment` を取得すると、常に { } が返されます。これは、空のリストです。

すべてのコメントの名前を変更する

VBA では、コメントを削除してから再度作成することにより、コメントに適用されている名前を変更できます。しかし、そのためにはコメントが添付されているセルの位置を把握している必要があります。**worksheet** はもとより、**used range** でもすべてのセルを検索してコメントがあるかどうかを確認するには、長い時間がかかります。したがって、このような方法ではなく、シートにあるすべてのコメントを取得します。

VBA では、コメントを削除する前にその **Parent** プロパティを取得しておくことで、再作成する場所を特定できます。VBA では Microsoft Excel、Word、PowerPoint のほぼすべてのオブジェクトに **Parent** プロパティがあります。しかし、Microsoft Office 2004 AppleScript では、どの Office アプリケーションでもこのプロパティはありません。このため、すべてのコメントを取得するには、使用されている範囲のすべてのセルをループする必要があります。VBA の方法は使用しません。次に VBA コードを示します。

```
Public Sub ChangeCommentName()
        Dim ws As Worksheet
        Dim cmt As Comment
        Dim sCmtText As String
        Dim sOldName As String
        Dim sNewName As String
        sNewName = "new name"
        sOldName = "old name"
        For Each ws In ActiveWorkbook.Worksheets
                For Each cmt In ws.Comments
                        With cmt
                                sCmtText = Application.Substitute( _
                                        .Text, sOldName, sNewName)
                                .Delete
                                .Parent.AddComment Text:=sCmtText
                        End With
                Next cmt
        Next ws
End Sub
```

次に AppleScript を示します。VBA とは大きく異なり、コメントではなくセル全体をループします。

```
tell application "Microsoft Excel"
        set newName to "new name"
        set oldName to "old name"

        repeat with ws in (get every worksheet in active workbook)
                set ur to used range of ws
                repeat with i from 1 to count cells of ur
                        set cl to cell i of ur
```

```
                        set cmt to Excel comment of cl
                        -- no error if nothing (dummy comment)
                        set vis to visible of cmt
                        --get any property , returns missing value if empty
                        if vis is not missing value then
                                set cmtText to Excel comment text cmt
                                set AppleScript's text item delimiters ¬
                                        to {oldName}
                                set chunks to text items of cmtText
                                set AppleScript's text item delimiters ¬
                                        to {newName}
                                set cmtText to chunks as Unicode text
                                delete cmt
                                add comment cl comment text cmtText
                        end if
                end repeat
                set AppleScript's text item delimiters to {""}
        end repeat
end tell
```

`repeat with cl in (cells of ur)` と `set cl to item i of (cells of ur)` はどちらも使用できません。代わりに、`repeat with i from 1 to (count cells of ur)` を使用する必要があります。この理由は、`count cells` は機能するものの、**cells** (または **every cell**) の取得は機能しないためです。その後で、`cell i of ur` を取得します。

AppleScript には、**Substitute** などのワークシート関数がありません。使用していないセルに **Substitute** などの関数を入力して、その値 (結果) を取得することもできますが、別の方法としては、組み込みの AppleScript `text item delimiters` を使用してテキストを置き換え、最後に既定の {""} に復元します。これは、有効な方法です。

ハイパーリンクを削除する

active sheet から、すべてのハイパーリンクを削除できます。VBA では次のようになります。

```
Public Sub DeleteActiveSheetHyperlinks()
        On Error Resume Next
        ActiveSheet.Hyperlinks.Delete
        On Error GoTo 0
End Sub
```

AppleScript では次のようになります。

```
tell application "Microsoft Excel"
        try
                delete every hyperlink of active sheet
        end try
end tell
```