Iterate backwards to delete rows

In AppleScript, if you don't iterate backwards to delete rows, it is easy to skip items and receive errors. Compare the following examples:

VBA

```
Public Sub DeleteEmptyRows()

Dim oTable As Table, oRow As Range, oCell As Cell, Counter As Long, _
NumRows As Long, TextInRow As Boolean

' Specify which table you want to work on.
Set oTable = Selection.Tables(1)
' Set a range variable to the first row's range
Set oRow = oTable.Rows(1).Range
NumRows = oTable.Rows.Count
Application.ScreenUpdating = False

For Counter = 1 To NumRows

          StatusBar = "Row " & Counter
          TextInRow = False

          For Each oCell In oRow.Rows(1).Cells
                   If Len(oCell.Range.Text) > 2 Then
                            'end of cell marker is actually 2 characters
                           TextInRow = True
                           Exit For
                  End If
          Next oCell

          If TextInRow Then
                   Set oRow = oRow.Next(wdRow)
          Else
                   oRow.Rows(1).Delete
          End If

Next Counter

Application.ScreenUpdating = True

End Sub
```

AppleScript

```
tell application "Microsoft Word"
         activate
          --Specify which table you want to work on
          set theTable to table 1 of selection -- or table 1 of active document
          set numRows to number of rows of theTable -- faster than counting rows
          set screen updating to false

          --iterate backwards because deleting!
          repeat with i from numRows to 1 by -1
                   set rowText to text object of row i of theTable

                  set status bar to "Row " & i
                  set textInRow to false

                   --you NEED the explicit gets if not setting variables!
                   repeat with theCell in (get every cell of rowText)
                            if (count of (get content of text object
                                    of theCell)) > 2 then
                                     -- end of cell marker is 2 characters,
                                     -- count faster than length
                                    set textInRow to true
                                     exit repeat -- no need to check other cells
                           end if
                  end repeat

                  if not textInRow then
                           delete row 1 of rowText
                  end if
          end repeat
          set screen updating to true
end tell
```

In the AppleScript example, note that the `repeat` loop has to be done differently, and backwards, because you are deleting items.
AppleScript does not have anything like **.Next(wdRow)** in Visual Basic for Applications (VBA), which allows you to move onto the next row no matter where you are in the loop, or whether you just deleted a row or not.

Repeat loops access objects ordered by index in a list

Any sort of `repeat` loop, even one iterating through a previously formed list, like the following example, is still composed of objects that are references:

```
set allRows to every row of theTable
```

In this case, as in many cases, the references are by index: {row 1 of table 1, row 2 of table 1, row 3 of table 1, etc.}. If you write the following code, hoping that it will fetch the `item i` of your list and return it as it was when you made the list, it will not:

```
set allRows to every row of theTable
repeat with i from 1 to (countallRows)
          set theRow to item i of allRows
```

List indices change as you delete objects in a list

The original item was a reference to `row i of theTable`. When it reevaluates `row i of the table`, it has different content than `row i` had previously, if you altered the table during the loop. The result is that it will skip an item.

This is not the case in an application like Microsoft Entourage where almost every application reference is to a "hard-coded" object with a unique **ID** in its database. That is a "luxury" of working with database applications. All references to objects, whether by name, by index, or other identifier, resolve to the "canonic" ID reference. For example, a list of `allMessages` in a folder consists of {message id 12345, message id 12346, message id 12347, message id 12348}. Even if you delete the second item in that list, it does not disappear from the list, so getting `item 3` of `allMessages` still gets you message id 12347, not id 12348.

In Microsoft Word, as in the Finder and most other applications, the index of the reference is reevaluated, and then an item is skipped. Later, you receive an error when AppleScript can't find the final indices because they no longer exist. Iterating backwards solves this problem, because indices lower than items you have deleted are not affected.

Using a unique identifier instead of an index

There are some ways that you can work around the propensity for Word to reevaluate every reference, such as using application references like **active document**.

For example, if you do the following, you lose your original reference, because Word reevaluates `theDoc` to the new document that moves to the front:

1. Set a variable, `theDoc`, to `active document`, which indicates the document in the front.
2. Minimize (**collapse**) `window 1` of the front document to the Dock.
3. Call `theDoc` again, perhaps to activate it.

You can get around this if you find something that uniquely identifies the current **active document** and refers to it by this identifier when setting the variable. The ideal unique identifier of any document is its **name**. No two documents can have the same name. You can set the variable by doing the following:

```
   tell application "Microsoft Word"
            set theName to name of active document
            set theDoc to document theName
end tell
```

Here the variable `theDoc` remains "hard-coded" to the currently active document. Even if you collapse the window to the Dock and another document becomes **active window**, the variable `theDoc` continues to point to the document that it was originally set to. Similarly, you can reactivate or do anything to the document, and the variable still points to the same document.

Iterate backward to prevent errors when deleting rows

However, in the example containing the list of `allRows`, each individual list item is a reference to a row in a list, ordered by index: {row 1 of table 1, row 2 of table 1 ,and so on}. The list gets reevaluated when called, resulting in skipped items and an error. To avoid this problem, you must iterate backwards. Meanwhile, there is no unique identifying feature for any row, at least not when getting **every row** as a list. It's the index for each row that constantly gets reevaluated as you delete rows, so iterating backwards is the only way to do it.

Note Although the dictionary definition for **table** states that you can get a **row** element "by name," that really means "by index" because rows do not have a **name** property.

Because of the `repeat with i from numRows to 1 by -1` syntax, in which `i` is a counter that does not need to be explicitly incremented by you, you also need an initializing statement in AppleScript that is equivalent to the following VBA statement:

```
Set oRow = oTable.Rows(1).Range
```

Those requirements are satisfied inside the following AppleScript repeat loop:

```
set rowText to text object of row 1 of theTable
```

Similarly, you don't need to check if `textInRow` is *true* because there's nothing to do if it is: the `i` counter will increment by itself.

The `screen updating` and **status bar** features work the same in AppleScript as in VBA, except that, with the adaptations above, you will see the status bar counting down to 1, which also shows you how far there is to go before it finishes.