Appendix 13 – Word Macro Techniques

(Version 17.02.21)

Why this appendix?

When I reached the age of 72, and knowing that no-one can live forever, I've thought about how best to pass on what I have learned during the previous 14 years of creating macros for editors. The macros themselves will still be here when I'm gone, but I'm aware that the repertoire of techniques I have in my brain will disappear unless I commit them to (electronic) paper.

What follows is a fairly random selection of ideas, sorry. It's quite difficult to systematise it because many of the techniques are linked, but this is definitely a work in progress. Indeed, as soon as I started to explain some of my ideas, I began to question them and also to wonder if it was possible to do various other things differently, and realised that they might prove useful in future macros, so that has meant that progress is quite slow.

If there are items in comment bubbles, then that's an indication that it's a bit that I'm at work on – a sort of 'Men at work' signboard. So please feel free to direct comments and/or questions about those bits – indeed, about any of the text below.

I hope that at least some of it proves useful to you.

Resources

Bibliography

1) (Definitely) *Macro Cookbook*, Jack Lyon (2012 – ISBN: 978-1-4341-0332-1)

2) (Probably) *An introduction to macro programming* Paul Beverley See: Appendix 13 – Word Macro Techniques

3) (Possibly) *Word 2007 Macros & VBA Made Easy*, Guy Hart-Davis (date? – ISBN: 978-0-07-161479-5)

4) (Reference) *Writing Word Macros*, Steven Roman (1999 – ISBN: 1-56592-725-7)

Four videos that might help:

Programming Word macros 1 (23:23)
An intro to the idea of programming Word macros
https://youtu.be/bivzgSTfbbk

Programming Word macros 2 (15:49)
Stepping through macros, watching what they do
https://youtu.be/igckZJ0euHk

Programming Word macros 3 (28:11)
Genesis of a macro – Part I
https://youtu.be/iGgBka7H-1w

Programming Word macros 4 (22:11)
Genesis of a macro – Part II
https://youtu.be/NWnmoRRUAKQ

...plus those videos mentioned in the sections below.

Changing things within specific paragraphs, sentences and words

(Videos: https://youtu.be/uwLmKZr07ws)

In my video, I go through examples of making changes to the content of paragraphs, sentences and words. The idea is if you want to "something" to certain elements of every paragraph (or sentence or word) in the whole document (or a selection thereof.)

These are the macros I used in the video:

```
Sub DIYFormatHeadwords()
```

```
Sub DIYColourLongSentences()
```

```
Sub DIYColourLongWords()
```

However, You might want to add this at the start of your macros:

```
' Check if user wants to work on whole file of selection
If Selection.End = Selection.Start Then
  myResponse = MsgBox("Do this to the WHOLE file?", _
       vbQuestion + vbYesNo)
  If myResponse = vbNo Then Exit Sub
  Set rng = ActiveDocument.Content
Else
  Set rng = Selection.Range.Duplicate
End If
```

In other words, of an area of text is selected, the macro will go ahead and make the changes, just to that area. However, if no text is selected, this code will alert you and ask if you *really* want to make the changes throughout the *whole* document.

'Find and do'

Commented [PB1]: Need to write this section

This is a very powerful concept: Look through a document

```
Sub DIYFindAshortMacro()
```

Counting the occurences of specific text

The 'Find and do' technique could be used as a way of counting the number of occurrences of something, but it would be very slow. Instead, you can use a technique based on find and replace. which you use to find the item you want to count and replace it by the same thing but with one extra character added. Here's a sample of the code:

```
myText = "hello"

' Find current length of file
myTot = ActiveDocument.Range.End

' Do token F&R
Set rng = ActiveDocument.Content
With rng.Find
  .ClearFormatting
  .Replacement.ClearFormatting
  .Text = myText
```

```
  .Replacement.Text = "^&!"
  .MatchCase = True
  .MatchWildcards = False
  .Execute Replace:=wdReplaceAll
End With
DoEvents
myCount = ActiveDocument.Range.End - myTot
If myCount > 0 Then WordBasic.EditUndo
MsgBox myCount
```

You record the current length of the file, do the F&R (the `"^&!"` means "that which you found, plus one extra character"), then find the increase in length of the file.

Finally, you undo the F&R, but ***only*** if the file is longer. If it found nothing, then it changed nothing, so the `WordBasic.EditUndo` would throw up a 'Nothing to undo' error.

For years, that was the fastest counting algorithm I could find. There is now a faster algorithm, but it only works for straightforward text counting. It can be used for both case-sensitive and non-case-insensitive counts, and it can be a whole-word count, but it can't be used for wildcard counting, such as that used in DocAlyse.

So, the faster technique is to text manipulation. You grab the whole of the text as a single string (yes, even if it's a 400,000-word book!), use `Len()` to find its length, use `Replace()` to make the same sort of change as you do with the F&R version, then find the new length of the string.

Here's a simplistic version of the code:

```
myText = "hello"

allText = ActiveDocument.Range.Text
totChars = Len(allText)
myCount = Len(Replace(allText, myText, myText & "!")) - totChars
```

But beware! This will count text-in-text. In other words, if you want to count the number of times 'etc' occurs, don't try it on a text like this:

"Visit an art shop to fetch some etchings and sketches, etc: a bottle of ketchup, a clump of vetch, a technical drawing of a valve and petcock next to a wetcell, a kingbird or petchary, an arrow from a fletchery, a Vietcong drinking a dietcoke or Sir Charles Sketchley receiving his baronetcy, etc. It'll make you feel tetchy and wretched, and you might even retch!"

The answer would not be 2, as you might expect, but 17!

So you need to 'prepare' your all-the-text string so that every single word has a space either side of it, and then in the final two lines, you can do the count – very quickly!

```
myText = "hello"

allText = " " & ActiveDocument.Content.Text & " "

' Prepare to change all punctuation to " "
' plus all "^p" to "^p " and all "^t" to "^t "

chs = " , . ! : ; [ ] { } ( ) / \ + "
' The variable 'chs' will hold all the Replace()
' items you want to make to the all-the-text string
chs = chs & ChrW(8220) & " "
chs = chs & ChrW(8221) & " "
chs = chs & ChrW(8201) & " "
chs = chs & ChrW(8222) & " "
```

```
chs = chs & ChrW(8217) & " "
chs = chs & ChrW(8216) & " "
chs = chs & ChrW(8212) & " "
chs = chs & ChrW(8722) & " "
chs = chs & vbCr & " "
chs = chs & vbTab & " "

' To force space at start; no space at end
' i.e. one space for each character that
' needs changing to a space
chs = " " & chs & " "
chs = Replace(chs, "  ", " ")
chs = Replace(chs, "  ", " ")

' Make all the replaces on the all-the-text string
chars = Split(chs, " ")
For i = 1 To UBound(chars)
  allText = Replace(allText, chars(i), " ")
Next i

' At last, we're ready to do the counting
totChars = Len(allText)

schText = " " & myText & " "
myCount = Len(Replace(allText, schText, schText & "!")) - totChars
```

Of course, the preparation work in the first stage of this code takes time, but the final two lines of code do the actual counting, so you can use these final two lines to count any and every word/phrase that you want to – on a 'whole-word' basis.

Manipulating the screen

If the selection point is not currently on screen, the following command brings the selection point to about 1/4 or 1/3 the way down the visible window.

```
ActiveWindow.ScrollIntoView Selection.range
```

Actually, I sometimes find it more helpful to bring the selection point right to the very top of the screen, so that I can easily find it (especially if the text size is small on screen), so I use my macro, *JumpScroll*:

```
Sub JumpScroll()
' Version 08.10.18
' Scrolls current line to the top of the page

Set rng = Selection.range.Duplicate
ActiveDocument.ActiveWindow.LargeScroll down:=1
rng.Select
ActiveDocument.ActiveWindow.SmallScroll down:=1
End Sub
```

The LargeScroll moves the selection point off screen, you reassert the selection point and a SmallScroll brings it to the top of the screen.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

If you're changing the selection point a lot, while the user doesn't need to see what's happening, you can save execution time by not allowing the screen to be updated.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
Application.ScreenUpdating = False/True
```

But beware that if the macro crashes, for some reason, while screen updating is off, you're in trouble! So you might want to add some error handling (q.v.), to switch screen-updating back on in the event of an error.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

You can change the size of the screen, but it fails if the window is either maximised or minimise, so normalise it first.

```
ActiveDocument.ActiveWindow.WindowState = wdWindowStateNormal
Application.Resize Width:=myWidth, Height:=myHeight
```

(Beware that some Macs don't support `Application.Resize`.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

But you can also find out what screen size is available and **then** change the size of your window...

1) To open a new file in the middle of the screen, at a specific distance from the edge of the screen (doesn't work on some Macs):

```
Sub OpenInMiddleScreen()

' User opens the chosen file
Dialogs(wdDialogFileOpen).Show

' Check how much screen area is available
scnHeight = Application.UsableHeight
scnWidth = Application.UsableWidth

' Do some calculations to decide on the window size,
' to leave a margin all the way around

mySideMargin = 100
myTopMargin = 50

Application.Move Left:=mySideMargin, Top:=myTopMargin

wdth = scnWidth - 2 * mySideMargin
ht = scnHeight - 2 * myTopMargin

' Resize the window
Application.Resize Width:=wdth, Height:=ht
End Sub
```

2) (Again, not on some Macs...) Open each new file with same size window as the current file, but further down and to the right. For this, you need to read the parameters of the current window and then open the new file.

```
Sub OpenDownAndRight()

myJump = 50

' Read the existing window parameters
nowWdth = Application.Width
nowHt = Application.Height
nowLeft = Application.Left
nowTop = Application.Top

' User opens the new file
```

```
Dialogs(wdDialogFileOpen).Show

newLeft = nowLeft + myJump
newTop = nowTop + myJump

' Set window's top left position
Application.Move Left:=newLeft, Top:=newTop

' How much space is available
scnHeight = Application.UsableHeight
scnWidth = Application.UsableWidth

' Do calculations to see if the window will go off screen...
wdth = nowWdth
rtMargin = scnWidth - newLeft - wdth

' and if so choose a better parameter for width
If rtMargin < 0 Then wdth = wdth + rtMargin

ht = nowHt
btmMargin = scnHeight - newTop - ht

' ... if so choose a better parameter for height
If btmMargin < 0 Then ht = ht + btmMargin

Application.Resize Width:=wdth, Height:=ht
End Sub
```

Counting the occurences of specific text

Dealing with track changes

When you want to do something **without** track changes, first see what the current state of TC is, then make the changes, then restore TC to as it was. Also, by using `doTrack = True/False`, you can set whether or not to do the changes with TC off:

```
doTrack = False ' Or True
revsView = ActiveWindow.View.RevisionsView
myTrack = ActiveDocument.TrackRevisions
If doTrack = False Then ActiveDocument.TrackRevisions = False

' Make the changes

Blah blah blah

' Restore TC
ActiveDocument.TrackRevisions = myTrack
ActiveWindow.View.RevisionsView = revsView
```

Select the whole of something

```
Selection.Expand wdParagraph
Selection.Expand wdTable
Selection.Expand wdSentence
Selection.Expand wdWord
```

But MS Word's idea of what constitutes a 'word' **includes** the apostrophe **and** the following space. In this sentence, I've highlighted some of the items that consitute a 'word':

> But MS Word's idea of what constitutes a 'word' **includes** the apostrophe **and** the following space. In this sentence, I've highlighted some of the 'word's.

The idea is that a 'word' is everything up to, but not including, the start of the next word (I deliberately put two spaces before 'includes'), and it includes the punctuation.

And the following sentence has eight 'word's, not five:

> Remember "double quotes" are included.

But if you just need the text of the word itself, then to avoid the following space(s) and/or the closing single quote, you could, in theory, use:

```
Selection.MoveEndWhile cset:=ChrW(8217) & " '", Count:=wdBackward
```

But don't!!!

I used this for years until I found that it was the culprit behind the enexplained crashes I was getting when I was using a macro in the region of a comment. Instead, use:

```
Do While InStr(ChrW(8217) & "' ", Right(Selection.Text, 1)) > 0
  Selection.MoveEnd , -1
  DoEvents
Loop
```

The DoEvents is added because if this bug in VBA does rear its ugly head, at least you'll be able to exit the macro cleanly, without actually crashing Word.

(If you're interested, the Cset command above means: move the selection backwards, past any collection of characters that include, space, straight apostrophe and curly close quote, i.e. curly apostrophe = `Chrw(8217)`.

If you're using a range, rather than a selection:

```
rng.Expand wdWord
```

Then use:

```
Do While InStr(ChrW(8217) & "' ", Right(rng.Text, 1)) > 0
  rng.MoveEnd , -1
  DoEvents
Loop
```

Other things you can select

We showed above that you can select a paragraph, table, word or sentence with

```
Selection.Expand
```

And you can do the same for a range:

```
rng.Expand wdParagraph
rng.Expand wdTable
```

```
rng.Expand wdSentence
rng.Expand wdWord
```

There are other things you can select, but only by using selection. So these commands allow you to define a range – for the currently active selection – giving the section, page, line, paragraph, table or table cell that the ***start*** of the selection is in. It does so ***without*** changing the selection, which can be quite useful.

You use:

```
set rng = ActiveDocument.Bookmarks("\Section").Range
set rng = ActiveDocument.Bookmarks("\Page").Range
set rng = ActiveDocument.Bookmarks("\Line").Range
set rng = ActiveDocument.Bookmarks("\Para").Range
set rng = ActiveDocument.Bookmarks("\Table").Range
set rng = ActiveDocument.Bookmarks("\Cell").Range
```

But if all you want to do is, say, select the whole of the current page, then it's

```
ActiveDocument.Bookmarks("\Page").Range.Select
```

Where am I?

Here are a few ideas about how to find where the cursor (Selection) or range has ended up.

Which page and line number is the cursor currently in?

```
pageNum = rng.Information(wdActiveEndAdjustedPageNumber)
lineNum = rng.Information(wdFirstCharacterLineNumber)
```

There are ways to find out where the

Is the cursor currently in a table? Yes or no.

```
inAtable = rng.Information(wdWithInTable)
```

Which paragraph (table) is the cursor currently in?

```
Set rng = ActiveDocument.range(0, Selection.End)
paraNum = rng.Paragraphs.Count

Set rng = ActiveDocument.range(0, Selection.End)
tableNum = rng.Tables.Count
```

(The latter code pair only tells you, if you aren't actually **in** a table, how many tables there **above** the cursor.)

One use of this is if you want to do something "from the current table onwards". Here's an example, which steps through the tables, one by one, and you can stop the macro if you get to one you want to edit. Then you just re-run the macro and carry on.

The there's information on which column/row the cursor is in:

```
   myColNum = rng.Information(wdStartOfRangeColumnNumber)
   myRowNum = rng.Information(wdStartOfRangeRowNumber)
```

I'm not sure what this is for. Have a play and let me know! :-)

```
MsgBox Selection.Information(wdHorizontalPositionRelativeToPage)
```

(List of `.Information` items is shown at the end of the file.)

```
Sub StepThroughTables()
' Version 21.11.18
' Steps through tables, one by one

Set rng = ActiveDocument.range(0, Selection.End)
tableNum = rng.Tables.Count
totTables = ActiveDocument.Tables.Count
For i = tableNum + 1 To totTables
  ActiveDocument.Tables(i).Select
  Selection.Collapse wdCollapseStart
  Set rng = Selection.range.Duplicate
  ActiveDocument.ActiveWindow.LargeScroll down:=1
  ActiveDocument.ActiveWindow.SmallScroll down:=1
  Selection.MoveUp wdParagraph, 1
  rng.Select
  Selection.MoveEnd wdWord, 1
  myResponse = MsgBox("Continue?", vbQuestion + vbYesNoCancel)
  If myResponse <> vbYes Then Exit Sub
Next i
Beep
End Sub
```

Problems during long-running macros

When you want to stop a macro running ("I didn't mean to run this particular macro!" or "This macro is taking too long!"), the theory is that you should be able to hold down the Ctrl key and press the Break key (unless you have a laptop that doesn't possess a Break key, as I have!) and the macro should stop.

However, when a macro is running that's very intensive, Word can get itself in a twist, and may totally ignore the Break key.

Worse still, if the user clicks on the screen, wondering if the macro has died, Word may well crash! I do try to warn people: "When a macro is running, **DON'T CLICK THE SCREEN!**" but it's an instinctive reaction when you're wondering what's going on. I know, I do it myself!

So, if you're going to run an intensive macro, and you fear that it might be long and tedious, open the Visual Basic first (use Alt-F11 – or if you then get a grey-only screen, go back to the Word file and use Alt-F8 and click Edit). Then move the VBA window so that you can see the top edge of this window. Why? (a) on that top line will be something like:

Microsoft Visual Basic for Applications - Chapter 04_PB - [NewMacros (Code)]

where "Chapter 04_PB" is the name of the file where the cursor is currently placed. Then when you run the macro, it changes to:

Microsoft Visual Basic for Applications - Chapter 04_PB [Running] - [NewMacros (Code)]

so you can see whether the macro is still running. If you want to stop the macro running, you can click the Reset (■) icon (like a DVD Stop icon) on the ribbon. In fact, you can click the Pause (‖) icon, which will take you into Debug mode, so that you can see where the macro has got to and then, if you decide it's OK and want to continue, click the Run (►) icon, or press F5.

What's more, if the macro uses different files then the title line tells you which file currently has the input focus, i.e. the cursor or the current `Selection`).

However, this way of halting a macro isn't 100% reliable. The VBA window itself does sometimes freeze – then you just have to crash Word and restart it. (You did remember to save the working file before running the macro, didn't you?!)

So it's worthwhile (I'd almost say essential) putting some `DoEvent` commands into the program at critical (busy) stages of its operation. This command doesn't actually do anything specific, but it seems to let the Word window 'catch up with' VBA, making it more likely that Ctrl-Break, or Reset (or Pause) will work, and you'll be able to exit cleanly from the macro.

Running other facilities from within a macro

You can run other macros from within a macro:

```
Application.Run MacroName:="AutoCurlyQuotesOFF"
Application.Run MacroName:="AutoListOff"
```

And you can run some of Word's functions, though I haven't found any rhyme or reason why some things work and others don't (answers on a postcard, please!). These two work:

```
Application.Run MacroName:="EditUndo"
Application.Run MacroName:="NextChangeOrComment"
```

but

```
Application.Run MacroName:="NavPaneSearch"
```

is a no-go, so for that, you have to use:

```
CommandBars("Navigation").Visible = True
```

However, if you use `Application.Run MacroName:="EditReplace"` then, after you've done your search, the *EditReplace* macro will still be running, and so when you close the F&R window, Word generates an error. So you have to instead use:

```
CommandBars("Menu Bar").Controls("Edit").Controls("Replace...").Execute
```

Either that, or you can add two error trapping lines:

```
On Error GoTo theEnd
Application.Run MacroName:="EditReplace"

theEnd:
End Sub
```

Also, you can sometimes use (for macros in the Normal template):

```
Call FRedit
Call AutoListOff
```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

You can open the Comments pane with:

```
ActiveDocument.ActiveWindow.View.SplitSpecial = wdPaneComments
```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Here are some examples of other things you can do with `Commandbars`. To change the width of the Navigation pane and the Styles pane:

```
Application.CommandBars("Styles").Width = 200
Application.CommandBars("Navigation").Width = 200
```

The above is put to good use in this macro:

```
Sub NavPaneCustomize()
' Version 09.01.21
' Opens the navigation pane where and how you want

h = 600
w = 400

' doSetUp = True
doSetUp = False

If doSetUp = True Then
  w = Application.CommandBars("Navigation").Width
  h = Application.CommandBars("Navigation").Height
  MsgBox "H: " & h & vbCr & vbCr & "W: " & w
  Exit Sub
End If

If Application.CommandBars("Navigation").Visible = False Then
  Application.CommandBars("Navigation").Visible = True
  'Application.CommandBars("Navigation").Position = msoBarRight
  Application.CommandBars("Navigation").Position = msoBarLeft
  'Application.CommandBars("Navigation").Position = msoBarFloating
  'Application.CommandBars("Navigation").Height = h
  Application.CommandBars("Navigation").Width = w
Else
  Application.CommandBars("Navigation").Visible = False
End If
End Sub
```

File handling

To find the name of the current file:

```
fileName = ActiveDocument.Name
fullFileName = ActiveDocument.FullName
```

These will give you, respectively, just the name of the file: “Chapter 04_PB.docx” or the whole address as well: “C:\MyFiles\WIP\MyCurrentBook\Chapter 04_PB.docx”

If you ask for the name of the current pane by using:

```
paneName = ActiveWindow.Caption
```

then you’ll probably get just the filename: “Chapter 04_PB” (without the ‘.docx’). However, if you’ve opened a second window on that file, you’ll get “Chapter 04_PB:01” or “Chapter 04_PB:02”.

If you want to close the current pane, to go back to a single view, use:

```
ActiveWindow.ActivePane.Close
```

Handling multiple files

I've written a range of different multi-file macros, so here I explain the code used to capture the names of the files in a folder, ready to "do something" with each of the files (or a sub-selection of those files).

```
            Check the number of documents currently open
docCount = Documents.Count
            Open the File Open dialogue box
Dialogs(wdDialogFileOpen).Show
            If the user has actually opened a document, then close it again
If Documents.Count > docCount Then ActiveDocument.Close
            Read the current directory
dirPath = CurDir()
            Point the filename reader, Dir(), to my directory
ChDir dirPath
            Use Dir() to read the name of the first file in this directory
            (you need the PathSeparator because it's different
            on Macs from PCs)
myFile = Dir(CurDir() & Application.PathSeparator)
            Create a new file for the list
Documents.Add
numFiles = 0
            As long as Dir() has found a file...
Do While myFile <> ""
            and if it's a Word-readable file (.doc, .docx or .rtf)...
  If InStr(LCase(myFile), ".doc") > 0 Or InStr(LCase(myFile), ".rtf") > 0 Then
            then enter its name into the list
    Selection.TypeText myFile & vbCr
            Count how many files there are
    numFiles = numFiles + 1
  End If
            Read the next file from the same directory
  myFile = Dir()
Loop
            Add the directory path at the top of the list
Selection.TypeText dirPath
            Read the directory name, but excluding the delimiter
Selection.MoveStartUntil cset:=":\", Count:=wdBackward
dirName = Selection
```

If you look in any of my multifile macros, you'll see that I also sort the list of files into alphabetical order. That's not necessary for PCs, but on Macs, the `Dir()` command doesn't pull up the files in alphabetical order for some reason.

Looking through the open windows/files

On the face of it, this is very straightforward. You can look through each of a set of open windows in order to find if there is a particular file that the macro is looking for. However, it is one bit of code that has caused real headaches, over the years.

Looking for the 'zzFReditList' file...

```
gottaDoc = False
For Each myWnd In Application.Windows
  thisName = myWnd.Document.Name
  If InStr(thisName, "zzF") > 0 Then
    gottaDoc = True
    myWnd.Document.Activate
```

```
    Beep
    Exit For
  End If
Next myWnd
```

And the other method:

```
gottaDoc = False
For Each thisDoc In Documents
  thisName = thisDoc.Name
  If InStr(thisName, "zzF") > 0 Then
    gottaDoc = True
    thisDoc.Activate
    Exit For
  End If
Next thisDoc
```

Just looking in my 'TheMacros' file, I see that I use the former method thirteen times and the latter six times. What I now **never** use is:

```
numDocs = Application.Documents.Count
For i = 1 To numDocs
  Set thisDoc = Application.Documents(i)
  thisName = thisDoc.Name
  If InStr(thisName, "zzF") > 0 Then
    thisDoc.Activate
    Exit For
  End If
Next i
```

There were times when it came up with the error:

> Run-time error '5941':
> The requested member of the collection does not exist.

And when I checked, I found that `numDocs` was, say four when, in fact, there were only three open Word files, so the fourth file it was looking for didn't exist. It wasn't a repeatable error – a programmer's nightmare!

So that's why I use the command: `For Each ... In`.

But remember that, with the `In Application.Windows` version, the user might have two or more windows open for each file. If that's crucial then you could use something like this (which I use in *FRedit*):

```
allFileNames = ""
For Each myDoc In Documents
  myFullName = myDoc.FullName
  If InStr(allFileNames, myFullName) = 0 Then
```

`...` do various things with this `myDoc`

```
    allFileNames = allFileNames & myFullName
  End If
Next myDoc
```

Copy text out into a new file

If you want to scrape the text out into a new file, perhaps so that you can analyse it without affecting the original, the natural thought would be to use copy and paste – a bad idea for all sorts of reasons! Instead, you can use this:

```
Set rng = ActiveDocument.Content
Documents.Add
Selection.FormattedText = rng.FormattedText
```

Or if you only want pure text and no formatting, use:

```
Set rng = ActiveDocument.Content
Documents.Add
Selection.Text = rng.Text
```

Copying into a new file in this way avoids using the clipboard, which is then available for other uses, if necessary, and it seems to be slightly quicker, but it's only fractions of a second. (It also avoids the error that used to annoy me when I closed Word: *"You placed a lot of content on the clipboard. Do you want this content to be available to other applications after you quit Word?"*)

Note that this only copies the main text, not text in end/footnotes or text boxes. If you want absolutely all the text, you could use my *CopyTextWithSomeFeatures* macro, though it doesn't give you the full formatting (styles etc.) that you get by using `FormattedText`, but rather just the pure text, plus a remembrance of bold, italic, super/subscript, etc.:

```
Call CopyTextWithSomeFeatures
```

Doing things in specified places/files

Commented [PB2]: Done as far as here.

You can do something in a specific named file. For example, to type some text at the top of a specific file:

```
Set rng = Documents("zzSwitchList.doc").Content
rng.InsertBefore Text:="Hello" & vbCr
```

but remember that if the file 'zzSwitchList.doc' is not open, the macro will give a 'Bad file name' error.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

If you have some text selected, you could do things at different places within it. For example, this extract finds the third paragraph within the selection, then makes the second word of the paragraph bold and the fourth character within that word big, selects it and then goes back to the original selection after you've pressed OK:

```
Set rngSel = Selection.range.Duplicate
Set rng = rngSel.Paragraphs(3).range
Set rng = rng.Words(4)
rng.Font.Bold = True
Set rng = rng.Characters(2)
rng.Font.Size = 40
rng.Select
MsgBox "Look!"
rngSel.Select
```

but beware that if it said `Set rng = rngSel.Paragraphs(3).range` when only two paragraphs were selected, it would generate an error and ditto if the numbers of words and characters are more than are available. But I'm just trying to demonstrate what different ranges and selections you can make.

Have you noticed that to set a range in **paragraphs**, you have to use `Paragraphs(30).range`, whereas for Words and Characters, you don't need the `.range`. I've no idea why! Each time I use these, I try with or without a `.range` and see if it errors (I can never remember).
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

And you can do sentences to:

```
Set rng = rngSel.Sentences(3)
```

And to highlight a selection or a range in a colour, or change font colour, use, for example:

```
Selection.range.HighlightColorIndex = wdGray25

Selection.range.Font.ColorIndex = wdBlue

' Make the third word of the selection red
Set rng = Selection.range.Duplicate
rng.Words(3).Font.Color = wdColorRed
```

(See below for explanation of `Color`/`ColourIndex`.)

Find and replace

To find and replace only within the selected area:

```
Set rng = Selection.range.Duplicate

With rng.Find
  .ClearFormatting
  .Replacement.ClearFormatting
  .Text = "cat"
  .Wrap = False
  .Replacement.Text = "dog"
  .Execute Replace:=wdReplaceAll
End With
```

You have to set .Wrap = False because if you were to use `.Wrap = wdFindContinue` it would F&R the whole of the document (well, the main text story).

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

If you do an F&R from within a macro, VBA assumes **only** the main story. So if you want to do the F&R in the foot(end)notes as well, use:

```
fnNum = ActiveDocument.Footnotes.Count
enNum = ActiveDocument.Endnotes.Count

For j = 1 To 3
  If j = 1 And fnNum = 0 Then j = 2
  If j = 2 And enNum = 0 Then j = 3
  Select Case j
    Case 1: Set rng = ActiveDocument.StoryRanges(wdFootnotesStory)
    Case 2: Set rng = ActiveDocument.StoryRanges(wdEndnotesStory)
    Case 3: Set rng = ActiveDocument.Content
  End Select
  DoEvents
  With rng.Find
    .ClearFormatting
    .Replacement.ClearFormatting
    .Text = myFind
    .Replacement.Text = myReplace
    .Wrap = wdFindContinue
    .Execute Replace:=wdReplaceAll
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
  End With
Next j
```

You can also extend the F&R to the text in text boxes, but this is more complex. FRedit has that facility, so if you need to F&R your textboxes, you'll need to pinch that section of code frpom FRedit.

Finding things – words

You can't use F&R to say "Find any one of the following bits of text", but if you're trying to find any one of a number of individual words, you can use something like this. It was aimed at the task of selecting the next conjunction, but it does a search of the selected text for "the next occurrence of any one of these words". If no text is selected, it searches from the cursor towards the end of the file.

```
Sub SearchTheseWords()
' Version 28.11.18
' Finds the next occurrence of any of a list of words

myWords = ":and:or:but:so:yet:if:"

Set rng = Selection.range.Duplicate
' If only a tiny selection...
If rng.Words.Count < 3 Then rng.Collapse wdCollapseEnd
' or nothing selected, search from cursor to the end of the file
If rng.Start = rng.End Then
  rng.End = ActiveDocument.Content.End
End If

myWords = ":" & myWords & ":"
For Each wd In rng.Words
  myTest = ":" & LCase(Trim(wd)) & ":"
  If InStr(myWords, myTest) > 0 Then
    wd.Select
    Exit For
  End If
Next wd
End Sub
```

Finding things – highlights/font attributes

The problem with searching for a particular font colour is that if your selection/range includes more than one colour, the colour number is reported as 9999999. So to be sure to find one colour only, you'd have to check every single character – slooow!

An alternative tactic is to check the colour of sections of text. If you do not get the answer 9999999, you know whether that section is/is not the colour you want. If it has mixed colours, then subdivide into smaller sections.

My technique is to check paragraphs, then words and then, if necessary, characters. So in this code section, we're looking for text in highlight colour `myHighlight` and then applying a strikethrough to that colour of highlighted text.

```
mixedColour = 9999999
For Each par In rng.Paragraphs
  col = par.range.HighlightColorIndex
  If col <> mixedColour Then
    If col = myHighlight Then par.range.Font.StrikeThrough = False
```

```
  Else
    For Each wd In par.range.Words
      col = wd.HighlightColorIndex
      If col <> mixedColour Then
        If col = myHighlight Then wd.Font.StrikeThrough = False
      Else
        For Each ch In wd.Characters
          col = ch.HighlightColorIndex
          If col <> mixedColour Then
            If col = myHighlight Then ch.Font.StrikeThrough = False
          End If
          DoEvents
        Next ch
      End If
      DoEvents
    Next wd
  End If
  DoEvents
Next par
```

Having watched a similar piece of code in action (using Selection, not rng), I could see that it was going quite slowly, so I had an idea for speeding it up: instead of using paragraph –> word –> character, I decided to use:

paragraph –> **sentence** –> word –> character.

Unfortunately, it fails miserably! If you want to see why, run this code segment:

```
ActiveDocument.Content.HighlightColorIndex = wdBrightGreen
For Each sn In ActiveDocument.Sentences
  sn.HighlightColorIndex = wdNoHighlight
Next sn
```

This should highlight the whole text and then remove the highlight from every single sentence in the document, shouldn't it? Well, you'll see that it leaves some text in green; it misses out some of the sentences, usually associated with ?, !, etc., and/or parentheses.

Or you could use this macro:

```
Sub HighlightAllSentences()
For Each sn In ActiveDocument.Sentences
  i = i + 1
  If i = 2 Then myColour = wdColorRed: i = 0
  If i = 1 Then myColour = wdColorBlue
  sn.Font.Color = myColour
  sn.Font.Bold = True
Next sn
End Sub
```

Nice idea, Paul (not)!

Later: I've just tried the following, and, unlike the `For Each` method, it works 100%!

```
ActiveDocument.Content.HighlightColorIndex = wdBrightGreen
For i = 1 To ActiveDocument.Sentences.Count
    ActiveDocument.Sentences(i).HighlightColorIndex = wdNoHighlight
  DoEvents
Next i
```

i.e. it does catch every single semtence.

OK, so I can try the '*paragraph* –> ***sentence*** –> *word* –> *character*' method again, provided I use the `For i = 1 To Whatever.Count`.

Here is a segment of code derived from what I've just written into FRedit. If you try it, it will find – in the fastest possible way – any text in bright green, amongst other colours of highlighting.

```
For Each par In rngNow.Paragraphs
  If par.range.HighlightColorIndex > 9999 Then
    For x = 1 To par.range.Sentences.Count
      If par.range.Sentences(x).HighlightColorIndex > 9999 Then
        For Each wd In par.range.Words
          If wd.HighlightColorIndex > 9999 Then
            For Each ch In wd.Characters
              If ch.HighlightColorIndex = fHiColour Then
                ch.Font.Emboss = True
              End If
            Next ch
          Else
            If wd.HighlightColorIndex = fHiColour Then
              wd.Font.Emboss = True
            End If
          End If
          DoEvents
        Next wd
      End If
      DoEvents
    Next x
  Else
    If par.range.HighlightColorIndex = fHiColour Then
      If Len(par.range.Text) > 1 Then par.range.Font.Emboss = True
    End If
  End If
Next par
fHiColour = wdBrightGreen
Set rngNow = ActiveDocument.Content
rngNow.Font.Emboss = False
For Each par In rngNow.Paragraphs
  If par.range.HighlightColorIndex > 9999 Then
    For x = 1 To par.range.Sentences.Count
      If par.range.Sentences(x).HighlightColorIndex > 9999 Then
        For Each wd In par.range.Sentences(x).Words
          If wd.HighlightColorIndex > 9999 Then
            For Each ch In wd.Characters
              If ch.HighlightColorIndex = fHiColour Then
                ch.Font.Emboss = True
              End If
            Next ch
          Else
            If wd.HighlightColorIndex = fHiColour Then
              wd.Font.Emboss = True
            End If
          End If
          DoEvents
        Next wd
      Else
        If par.range.Sentences(x).HighlightColorIndex = fHiColour Then
          par.range.Sentences(x).Font.Emboss = True
```

```
        End If
      End If
      DoEvents
    Next x
  Else
    If par.range.HighlightColorIndex = fHiColour Then
      If Len(par.range.Text) > 1 Then par.range.Font.Emboss = True
    End If
  End If
Next par
```

In case you're interested to know, this code is rarely used by FRedit because it's only required if the user is asking to change something in one highlight colour into a different highlight colour. For example, here the user is asking to leave most of the cats alone and only change those cats that are highlighted in green.

cat|dog

Because you can't do an F&R for a specific colour of highlight (only whether highlighting is ON or OFF), I use the above code to add an Emboss attribute to all the text that's in a particular highlight colour. Then I can do the F&R with Emboss as one Find characteristic, and then later remove all the Embossing. (This is explained in the section below about F&R.)

More about F&R

If you are doing F&Rs using `Selection`, and you want to preserve the original Find and Replace values after the macro has finished then use, for example:

```
oldFind = Selection.Find.Text
oldReplace = Selection.Find.Replacement.Text

With Selection.Find
  .ClearFormatting
  .Replacement.ClearFormatting
  .Text = "this"
  .Replacement.Text = "that"
  .Execute Replace:=wdReplaceAll
End With

Selection.Find.Text = oldFind
Selection.Find.Replacement.Text = oldReplace
```

However, if you only use ranges, such as:

```
Set rng = ActiveDocument.Content
With rng.Find
  .ClearFormatting
  .Replacement.ClearFormatting
  .Text = "this"
  .Replacement.Text = "that"
  .Execute Replace:=wdReplaceAll
End With
```

then you should find that the original Find and Replace value are retained (well, they are with Word 2010).

Search and destroy (joke!)

The following is a dummy macro that I use time and again. It sets up a search for something and then repeatedly looks for that 'thing' and, if it finds it, it does something to it, and then looks to see if there's another occurrence, but if there are no more of them, it stops.

First set a range:

```
Set rng = ActiveDocument.Content
```

or use...

```
Set rng = Selection.range.Duplicate
```

then...

```
' Go and find the first occurrence
With rng.Find
  .ClearFormatting
  .Replacement.ClearFormatting
  .Text = "thing"
  .Wrap = wdFindStop
  .Replacement.Text = ""
  .Forward = True
  .MatchWildcards = False ' Set as required
  .MatchWholeWord = False
  .MatchSoundsLike = False
  .Execute
End With

myCount = 0 ' Set as required
Do While rng.Find.Found = True
  myCount = myCount + 1

' Note where the end of the found item is
  endNow = rng.End

' Do various things with this "thing" it has found

' Be sure you're past the previous occurrence
  rng.End = endNow
  rng.Collapse wdCollapseEnd

' Go and find the next occurence (if there is one)
  rng.Find.Execute
Loop
MsgBox "Changed: " & myCount
```

Changing attributes of a selection by F&R

If you want to change various font attributes in a selection with F&R, you can specify them as follows.

```
Set rng = Selection.range.Duplicate
With rng.Find
  .ClearFormatting
  .Replacement.ClearFormatting
  .Text = "i"
  .MatchCase = False
```

```
  .Wrap = False
  .Replacement.Text = "^&"
  .Replacement.Font.Size = 20
  .Replacement.Font.Color = wdColorRed
  .Replacement.Font.StrikeThrough = True
  .Replacement.Font.Underline = True
  .Replacement.Highlight = True
  .Execute Replace:=wdReplaceAll
End With
```

Note that I used `.Replacement.Font.Color = wdColorRed` which is the colour as set by hexadecimal (number to base 16) values, in this case, `wdColorRed` is 000000FF. I could have used: `.Replacement.Font.ColorIndex = wdRed`, in this case, `wdRed` has the value 6.

Changing **highlight colour**s using F&R is different. Although you can give the font colour a value in an F&R, such as wdColorRed, highlighting is only ever True or False. If True, then it will appear in the currently selected highlight colour – whatever that happens to be. So if want to use a specific highlight colour with F&R, you need to memorise the current highlight colour, change the colour, use it and then, before you exit the macro, restore the original highlight colour:

```
oldColour = Options.DefaultHighlightColorIndex
Options.DefaultHighlightColorIndex = wdBrightGreen

' Do your find and replace here

Options.DefaultHighlightColorIndex = oldColour
```

Reading the font colour

For the font colour of the current selection:

```
myColourIndex = Selection.range.Font.ColorIndex
myColour = Selection.range.Font.Color
```

The first gives the simple colour number, so red is 6, and green is 11. The second is actually a hex number, so to see it easily (meaningfully), use `Hex(myColour)`, for which, red is 000000FF, and green is 0050B000 and black is FF000000.

If the selection includes more than one colour, both Color and ColorIndex give the seven-digit value 9999999 (a meaningless 98967F in hex).

For the colour used by the **style** within the selection:

```
paraColourIndex = ActiveDocument.Styles(Selection.range.Style).Font.ColorIndex
paraColour = ActiveDocument.Styles(Selection.range.Style).Font.Color
```

The following macro checks a selection, showing if another colour has been applied to the basic font colour of that style, and whether there is a mix of colours:

```
Sub FontColourReader()
' Version 21.11.18
' Reads style font colour + any applied colour

myMix = 9999999
paraColourIndex = ActiveDocument.Styles(Selection.range.Style).Font.ColorIndex
paraColour = ActiveDocument.Styles(Selection.range.Style).Font.Color
```

```
myColourIndex = Selection.range.Font.ColorIndex
myColour = Selection.range.Font.Color

myMessage = ""
myMessage = myMessage & "Style font colour = " & Hex(paraColour) & vbCr
If myColour = myMix Then
  myMessage = myMessage & "Mixed colours" & vbCr
Else
  If paraColour = myColour Then
    myMessage = myMessage & "No applied colour" & vbCr
  Else
    myMessage = myMessage & "Applied colour = " & Hex(myColour) & vbCr
  End If
End If
MsgBox myMessage

myMessage = ""
myMessage = myMessage & "Style font colour = " & paraColourIndex & vbCr
If myColourIndex = myMix Then
  myMessage = myMessage & "Mixed colours" & vbCr
Else
  If paraColourIndex = myColourIndex Then
    myMessage = myMessage & "No applied colour" & vbCr
  Else
    myMessage = myMessage & "Applied colour = " & myColourIndex & vbCr
  End If
End If
MsgBox myMessage
End Sub
```

Reading the font name and size

This macro checks a selection, showing if another font name or font size has been applied to the basic font name and size of that style, and whether there is a mix of names/sizes. The font size returned for a mix is, again, 9999999, but for a selection of mixed font names, it returns a null string, "":

```
Sub FontNameAndSizeReader()
' Version 21.11.18
' Reads style font Name + any applied Name

myMixName = ""
paraName = ActiveDocument.Styles(Selection.range.Style).Font.Name
myName = Selection.range.Font.Name

myMessage = ""
myMessage = myMessage & "Style font name = " & paraName & vbCr
If myName = myMixName Then
  myMessage = myMessage & "Mixed names" & vbCr
Else
  If paraName = myName Then
    myMessage = myMessage & "No applied name" & vbCr
  Else
    myMessage = myMessage & "Applied name = " & myName & vbCr
  End If
End If
MsgBox myMessage

myMix = 9999999
paraSize = ActiveDocument.Styles(Selection.range.Style).Font.Size
```

```
mySize = Selection.range.Font.Size

myMessage = ""
myMessage = myMessage & "Style font Size = " & paraSize & vbCr
If mySize = myMix Then
  myMessage = myMessage & "Mixed Sizes" & vbCr
Else
  If paraSize = mySize Then
    myMessage = myMessage & "No applied size" & vbCr
  Else
    myMessage = myMessage & "Applied size = " & mySize & vbCr
  End If
End If
MsgBox myMessage
End Sub
```

Information about styles

For a description of a given style use, say, `ActiveDocument.Styles("Heading 1").Description`, and the style can also be specified as, say, `wdStyleHeading1`, as here:

```
nStyle = ActiveDocument.Styles(wdStyleNormal).Description & vbCr & vbCr
H1Style = ActiveDocument.Styles(wdStyleHeading1).Description & vbCr
Selection.TypeText Text:="Normal style: " & nStyle
Selection.TypeText Text:="Heading 1: " & H1Style
```

The result, from this file, is:

Normal style: Font: (Default) Times New Roman, 11 pt, Left
Line spacing: single, Widow/Orphan control, Style: Quick Style

Heading 1: Font: (Default) Arial, 22 pt, Bold, Kern at 16 pt, Space
Before: 18 pt
After: 3 pt, Keep with next, Level 1, Style: Linked, Quick Style
Based on: Normal
Following style: Normal

Applying shading, foreground and background colours

I confess to not knowing what the difference is between foreground and background colours, but have a play and see what you can work out!

```
' Add 10% grey tinted background
Selection.Shading.Texture = wdTexture10Percent
' Back to no tint
Selection.Shading.Texture = wdTextureNone

' Yellow background
Selection.Shading.BackgroundPatternColor = wdColorYellow
' Return to no background
Selection.Shading.BackgroundPatternColor = wdColorAutomatic

' Yellow foreground (has a remarkably similar effect!)
Selection.Shading.ForegroundPatternColor = wdColorYellow
' This makes it black!
Selection.Shading.ForegroundPatternColor = wdColorAutomatic
' This makes it white
Selection.Shading.ForegroundPatternColor = wdColorWhite
```

```
' Also available for clearing background
Selection.Shading.Texture = wdTextureNone

' For some weird effects, try this
With Selection.Shading
 .Texture = wdTextureDarkDiagonalCross
 .ForegroundPatternColorIndex = wdBlue
 .BackgroundPatternColorIndex = wdYellow
End With
```

Beep and double-beep!

Doing a `beep` is obvious, but it's sometimes useful to give the user a double-beep. For example, SpellingSuggest gives a single beep if the word is spelt correctly, but a double-beep if it's a spelling error and has therefore been corrected. You may need to increase the delay time (0.2) if your system gives two beeps that just sound like one.

```
Beep
myTime = Timer
Do
Loop Until Timer > myTime + 0.2
Beep
```

Timing things

In order to time some process, say an analysis, you record the time at the beginning:

```
timeStart = Timer
```

and then at the end:

```
totTime = Timer - timeStart
```

If you want to give the user a chance to not show how long it took then put, at the beginning,

`showTime = True` or `False`

and then

```
If showTime = True And totTime > 60 Then
  MsgBox ((Int(10 * totTime / 60) / 10) & _
       "  minutes")
End If
```

Setting up arrays

You can set up an array of words (or phrases) so that you can do things with them, one by one. In this case, it's just displaying the words to the user, but there will obviously be more interesting applications!

```
allWords = ",red,blue,green"
myWord = Split(allWords, ",")
numWords = UBound(myWord)
For i = 1 To numWords
  MsgBox myWord(i)
Next i
```

The comma is what is called the ‘list separator’, but note too that there is a comma in front of my first word. That’s because arrays were invented by programmers and they think that number ranges should start with zero! So if the list was:

```
allWords = "black,red,blue,green"
```

then you would find that `myWord(0)` (that’s zero, not capital ‘O’) has the value `"black"`.

Note that the character used as a list separator can be anything you like. In the following, there are commas in the text, so I’ve used the vertcal bar (as used in FRedit):

```
allWords = "|fish, chips, and peas|ham, egg, and chips|pie, mash, and beans"
myWord = Split(allWords, "|")
numWords = UBound(myWord)
For i = 1 To numWords
  MsgBox myWord(i)
Next i
```

User input

We’ve already used, say, `MsgBox myWord(i)` as a way of displaying something, but the only input the user can make is OK. If you press the Escape key, it has exactly the same effect of making the macro continue on the line after the MsgBox.

Another more flexible example of using MsgBox is:

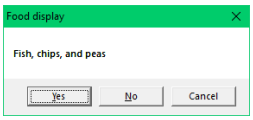
```
myResponse = MsgBox(myWord(i), vbOKCancel, "Food display")
If myResponse <> vbOK Then Exit Sub
```

So clicking Cancel or pressing the Escape key stops the macro.

And (using the example from the arrays section) you can also offer a No option:

```
allWords = "|Fish, chips, and peas|Ham, egg, and chips|Pie, mash, and beans"
myWord = Split(allWords, "|")
For i = 1 To numWords
  myResponse = MsgBox(myWord(i), vbYesNoCancel, "Food display")
  If myResponse = vbNo Then Beep
  If myResponse = vbCancel Then Exit Sub
Next i
```

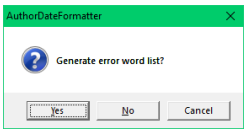
Here’s the output:



Here’s another example

```
myResponse = MsgBox("Generate error word list?", vbQuestion _
        + vbYesNoCancel, "AuthorDateFormatter")
If myResponse = vbCancel Then Exit Sub
If myResponse = vbYes Then generateList = True
```

Here’s the output:



If you want to input some text, you could use, say:

```
mySurname = InputBox("Surname?", "My naming macro")
```

If the user just presses Enter without typing in a name, or presses Escape or clicks Cancel, then myInput is a zero-length string, "".

Here's the output:

If you want to input a number, you could use, say:

```
myInput = InputBox("Option number?", "Macro name")
myNumber = Val(myInput)
```

If the user just presses Enter without typing in a number, or presses Escape or clicks Cancel, then myNumber has the value zero.

You can also use a default value to offer the user, such as the previous option chosen by the user:

```
myDefaultValue = "Whatever!"

myRequirement = InputBox("What do you want?", "Macro name", myDefaultValue)
```

String handling techniques

The following code isn't rocket science:

```
myText = "My sample string"

myLeft = Left(myText, 2)
myRight = Right(myText, 4)
myMiddle = Mid(myText, 4, 6)

For i = 1 To Len(myText)
  myChar = Mid(myText, i, 1)
  Debug.Print myChar
Next
```

I'm sure you'll be able to work out that the three commands generate: 'My', 'ring' and 'sample'.

Then the loop goes through the individual characters and displays them one by one in VBA's 'Immediate mode' area. (Press Ctrl-G in VBA to open the Immediate mode window.)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

To select the part of a string after a specific character, you can use `InStr`.

This example extracts the text inside the parentheses:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
parenOpen = InStr(mySample, "(")
myText = Mid(mySample, parenOpen + 1)

parenClose = InStr(myText, ")")
myText = Left(myText, parenClose - 1)
```

Or you could do it like this.

```
parenOpen = InStr(mySample, "(")
parenClose = InStr(mySample, ")")

myLen = parenClose - parenOpen
myText = Mid(mySample, parenOpen + 1, myLen - 1)
```

The thing you have to be careful of is what will happen if one of the characters is missing. If there's no close parenthesis, `parenClose` comes out as zero, so you get an 'Invalid procedure call or argument' in both cases.

Using the Like function

This function allows some flexibility in comparing strings. You have '?' to mean a single character, and '*' to mean any text.

```
myTest = "?ome*g"

Dim myText(4) As String
myText(1) = "Something"
myText(2) = "something"
myText(3) = "somewhat"
myText(4) = "homecoming"

For i = 1 To 4
  If myText(i) Like myTest Then
    MsgBox myText(i) & " is a match"
  Else
    MsgBox myText(i) & " is NOT a match"
  End If
Next i
```

For this test, all three will be a match, except 'somewhat'.

Probably more helpful are tests as in this example:

```
Do
  myInput = InputBox("Enter some text")

  If myInput Like "[A-Z]" Then
    MsgBox "Single uppercase letter"
  End If

  If Left(myInput, 3) Like "[A-Z][a-z]*" Then
    MsgBox "Looks like a word with an initial capital"
  End If

  If myInput Like "[yY]*" Then
    MsgBox "Yes!!"
  End If

  If myInput Like "[Nn]*" Then
    MsgBox "No way!!"
```

```
  End If

  If myInput Like "#" Then
    MsgBox "Single digit"
  End If

  If myInput Like "[0-9]" Then
    MsgBox "Single digit"
  End If
Loop Until myInput = ""
```

I think these should be reasonably self-explanatory. Note that '`#`' has the same meaning as '`[0-9]`'

Error handling

To switch error handling on:

```
On Error GoTo ReportIt
```

Then at the end of the macro, you can use something like this:

```
Exit Sub
ReportIt:
If Err.Number = 5174 Then
  MsgBox ("Couldn't find file: " & myFileName)
Else
  On Error GoTo 0
  Resume
End If

End Sub
```

What happens here is that when an error occurs, if it's a File Not Found error (which is the type of error that can be detected by testing whether Err.Number = 5174), you report that to the user, telling the which file it couldn't find, and then exit the macro.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

I'm never quite sure what `On Error Resume Next` does, but let's look at an example:

(Comment from the ever-helpful Howard Silcock: *When you include this statement, if any subsequent statement generates an error, then the error is ignored and execution proceeds to the next statement. This remains in force until you include a statement such as On Error Goto 0, which resets to the normal behaviour.*)

We're searching of a specific file, `tryThisName`, and if it's not found we report back to the user.

```
On Error Resume Next
                  If an error occurs, keep going on to this next line...
If tryThisName <> "" Then
                  Try to open a file of this name...
  Documents.Open tryThisName
                  If there isn't one, it errors
  If Err.Number = 5174 Then
                  Tell the user you couldn't find it
    MsgBox ("Can't find file: " & tryThisName)
                  Clear the error condition
    Err.Clear
  Else
                  If we did find a file, switch the error reporting off, just in case some
                  other error occurs...
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
    On Error GoTo 0
                Now we can carry on as normal...
    Application.Resize Width:=myWidth, Height:=myHeight
    Set wasSelected = Selection.range.Duplicate
    Selection.HomeKey Unit:=wdStory
    With Selection.Find
    etc. etc. etc.

  End If
End If
```

If application visibility is being switched off, you do need an error handler to switch it back on because all the Word windows will be invisible, if not!

```
On Error GoTo ReportIt
```

and then...

```
ReportIt:
Application.Visible = True
On Error GoTo 0
Resume
End Sub
```

Similarly, if you're switching ScreenUpdating off, to speed up execution, then add an error-handler to switch it back on again.

```
On Error GoTo ReportIt
```

and then...

```
ReportIt:
Application.ScreenUpdating = True
On Error GoTo 0
Resume
End Sub
```

Random text

For five paragraphs of 10 'Latin' sentences each, just type:

=lorem(5,10)

Here's an example paragraph of 10 sentences:

> Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna. Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci. Aenean nec lorem.

And or five paragraphs of 10 'English' sentences each, use:

=rand(5,10)

Here's an example paragraph of just four sentences (they're longer than the Latin ones):

> On the Insert tab, the galleries include items that are designed to coordinate with the overall look of your document. You can use these galleries to insert tables, headers, footers, lists, cover pages, and other document building blocks. When you create pictures, charts, or diagrams, they also coordinate with your current document look. You can easily change the formatting of selected text in the document text by choosing a look for the selected text from the Quick Styles gallery on the Home tab.

And here's another one, sent in by Ken Endacott:

=rand.old(5,10)

Here's an extract of what it produces:

> The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.

And I have them in my MultiSwitch list as:

ll
=lorem(5,10)

ee
=rand(5,10)

Really useful, if you need some text to play with.

All sorts of information

There's lots of information available from `Selection.Information()`

e.g. `Selection.Information(wdInCommentPane)`

(Confession time! I found this on the internet ages ago, but I can't acknowledge the source as I can't now find where it came from, sorry. It said it was for Word 2003, but the ones I've tried have worked OK.)

`wdActiveEndAdjustedPageNumber` returns the number of the page that contains the active end of the specified selection or range. If you set a starting page number or make other manual adjustments, returns the adjusted page number (unlike `wdActiveEndPageNumber`).

`wdActiveEndPageNumber returns` the number of the page that contains the active end of the specified selection or range, counting from the beginning of the document. Any manual adjustments to page numbering are disregarded (unlike `wdActiveEndAdjustedPageNumber`).

`wdActiveEndSectionNumber returns` the number of the section that contains the active end of the specified selection or range.

`wdAtEndOfRowMarker returns` True if the specified selection or range is at the end-of-row mark in a table.

`wdCapsLock returns` True if Caps Lock is in effect.

`wdEndOfRangeColumnNumber returns` the table column number that contains the end of the specified selection or range.

`wdEndOfRangeRowNumber returns` the table row number that contains the end of the specified selection or range.

wdFirstCharacterColumnNumber returns the character position of the first character in the specified selection or range. If the selection or range is collapsed, the character number immediately to the right of the range or selection is returned (this is the same as the character column number displayed in the status bar after "Col").

wdFirstCharacterLineNumber returns the character position of the first character in the specified selection or range. If the selection or range is collapsed, the character number immediately to the right of the range or selection is returned (this is the same as the character line number displayed in the status bar after "Ln").

wdFrameIsSelected returns True if the selection or range is an entire frame or text box.

wdHeaderFooterType returns a value that indicates the type of header or footer that contains the specified selection or range, as shown in the following table.

wdHorizontalPositionRelativeToPage returns the horizontal position of the specified selection or range; this is the distance from the left edge of the selection or range to the left edge of the page measured in points (1 point = 20 twips, 72 points = 1 inch). If the selection or range isn't within the screen area, returns −1.

wdHorizontalPositionRelativeToTextBoundary returns the horizontal position of the specified selection or range relative to the left edge of the nearest text boundary enclosing it, in points (1 point = 20 twips, 72 points = 1 inch). If the selection or range isn't within the screen area, returns - 1.

wdInClipboard For information about this constant, consult the language reference Help included with Microsoft Office Macintosh Edition.

wdInCommentPane returns True if the specified selection or range is in a comment pane.

wdInEndnote returns True if the specified selection or range is in an endnote area in print layout view or in the endnote pane in normal view.

wdInFootnote returns True if the specified selection or range is in a footnote area in print layout view or in the footnote pane in normal view.

wdInFootnoteEndnotePane returns True if the specified selection or range is in the footnote or endnote pane in normal view or in a footnote or endnote area in print layout view. For more nformation, see the descriptions of wdInFootnote and wdInEndnote in the preceding paragraphs.

wdInHeaderFooter returns True if the selection or range is in the header or footer pane or in a header or footer in print layout view.

Value	Type of header or footer
−1	None (the selection or range isn't in a header or footer)
0	Even page header
1	Odd page header (or the only header, if there aren’t odd and even headers)
2	Even page footer
3	Odd page footer (or the only footer, if there aren’t odd and even footers)
4	First page header
5	First page footer

wdInMasterDocument returns True if the selection or range is in a master document (that is, a document that contains at least one subdocument).

`wdInWordMail returns` True if the selection or range is in [as the original text of this item read...] *the header or footer pane or in a header or footer in print layout view.* [That's obviously someone's copy and paste from the earlier item on wdInHeaderFooter! I don't know what it *should* say as I do all my email on an old-fashioned Acorn Computers email system. Any ideas for a corrected text here, please?]

Value	Location
0	The selection or range isn't in an email message.
1	The selection or range is in an email message you are sending.
2	The selection or range is in an email you are reading.

`wdMaximumNumberOfColumns returns` the greatest number of table columns within any row in the selection or range.

`wdMaximumNumberOfRows returns` the greatest number of table rows within the table in the specified selection or range.

`wdNumberOfPagesInDocument returns` the number of pages in the document associated with the selection or range.

`wdNumLock returns` True if Num Lock is in effect.

`wdOverType returns` True if Overtype mode is in effect. The Overtype property can be used to change the state of the Overtype mode.

`wdReferenceOfType returns` a value that indicates where the selection is in relation to a footnote, endnote, or comment reference, as shown in the following table.

Value	Description
−1	The selection or range includes but isn't limited to a footnote, endnote, or comment reference.
0	The selection or range isn't before a footnote, endnote, or comment reference.
1	The selection or range is before a footnote reference.
2	The selection or range is before an endnote reference.
3	The selection or range is before a comment reference.

`wdRevisionMarking returns` True if change tracking is in effect.

`wdSelectionMode returns` a value that indicates the current selection mode, as shown in the following table.

Value	Selection mode
0	Normal selection
1	Extended selection ('EXT' appears on the status bar)
2	Column selection. ('COL' appears on the status bar)

`wdStartOfRangeColumnNumber returns` the table column number that contains the beginning of the selection or range.

`wdStartOfRangeRowNumber returns` the table row number that contains the beginning of the selection or range.

`wdVerticalPositionRelativeToPage returns` the vertical position of the selection or range; this is the distance from the top edge of the selection to the top edge of the page measured in points (1 point = 20 twips, 72 points = 1 inch). If the selection isn't visible in the document window, returns −1.

`wdVerticalPositionRelativeToTextBoundary returns` the vertical position of the selection or range relative to the top edge of the nearest text boundary enclosing it, in points (1 point = 20 twips, 72 points = 1 inch).

This is useful for determining the position of the insertion point within a frame or table cell. If the selection isn't visible, returns −1.

`wdWithInTable returns` True if the selection is in a table.

`wdZoomPercentage returns` the current percentage of magnification as set by the Percentage property.

Example

This example displays the current page number and the total number of pages in the active document.

```
MsgBox "The selection is on page " & _
    Selection.Information(wdActiveEndPageNumber) & " of page " _
    & Selection.Information(wdNumberOfPagesInDocument)
```

If the selection is in a table, this example selects the table.

```
If Selection.Information(wdWithInTable) Then _
    Selection.Tables(1).Select
```

This example displays a message that indicates the current section number.

```
Selection.Collapse Direction:=wdCollapseStart
MsgBox "The insertion point is in section " & _
    Selection.Information(wdActiveEndSectionNumber)
```

[ignore] FRedit list for formatting listings
`~^t*^13|^&`

Genesis of a macro

What I want to do here is the narrate what happened as I wrote a macro, hoping that the process itself will illustrate various techniques. Here goes...

The issue was that someone had seen my *NumbersToText* macro, which can whizz along through the text and, when it finds a number as figures, e.g. '342 soldiers', it changes it to 'three hundred and forty-two soldiers', and they asked if there was a macro that would change 'three hundred and forty-two soldiers' into '342 soldiers'.

The answer was no, but there is now: *TextToNumber*. So here's the story.

Analyse the problem

The first stage was to think about all the ways in which numbers up to 999 only. But that illustrates the problem of writing up about macros. As soon as I wrote that sentence, I realised that I really ought to add 1000! I'll do it when I've finished the write-up.) Here's some rubbish text. THe b;lue are the normal ways (in UK) that we express numbers in words, and the pink are a few 'funnies', which includes the American way of expressed numbers. (Sorry, no offence to my transatlantic cousins intended!)

> Sample here is three bits of text with forty things. But sixteen then three hundred and forty-two soldiers a hundred and forty-two marched up the hill, followed by a hundred boy scouts and one hundred and sixteen girl guides three hundred only is it now things like he had forty two soldiers seventy-nine and American three hundred

forty-two Ah, but what about two hundred and six but what about two hundred and sixteen but what about two hundred and forty. No chance with four hundred two, I suppose?

As you can see, it's going to be a quite complex process writing this macro.

List all the possible words

```
myUnits = ":one:two:three:four:five:six:seven:eight:nine:ten"
myTens = ":ten:twenty:thirty:forty:fifty:sixty:seventy:eighty:ninety:hundred"
myTeens = ":eleven:twelve:thirteen:fourteen:fifteen:sixteen:seventeen:
eighteen:nineteen"
allNumberWords = myUnits & myTens & myTeens & ":a:and:-:"
```

I've put them all in an order, so that 1–10 and the first ten words, so for a given word, say 'three', I can find it in the list, then if I count the number of colons to the left (3), I've converted from a word to a number.

Then 11–20 are the tens, to 'thirty' will generate 13, so I can subtract 10 and multiply by 10 to get the 30.

Then 21–29 are the teens (you can do the calculation as your homework!), and the final odds and ends are also needed, including the hyphen, which is a 'word' on its own as far as VBA is concerned.

Pick up the words into an array

To start simply, I assumed that the cursor was in the first word:

```
Set rng = Selection.range.Duplicate
rng.Expand wdWord
rng.MoveEnd wdWord, 8
Dim wd(8) As String
For i = 1 To 8
  thisWord = Trim(rng.Words(i))
  If InStr(allNumberWords, ":" & thisWord & ":") > 0 Then
    wd(i) = thisWord
    Debug.Print thisWord & " ";
  Else
    numWords = i - 1
    Exit For
  End If
Next i
```

The important learning elements are that the range, `rng`, starts as a zero-length range at the cursor.

```
Set rng = Selection.range.Duplicate
```

Then we expand it to the whole of the current word

```
rng.Expand wdWord
```

and then we load up the array, `wd()`, with the next eight words, one at a time:

```
thisWord = Trim(rng.Words(i))
```

trimming off the trailing space.

Then if thisWord, with a colon added on each end, is in allNumberWords

```
If InStr(allNumberWords, ":" & thisWord & ":") > 0 Then
```

Add it into the array

```
wd(i) = thisWord
```

And so we can see what's happening, we print it into the Immediate mode window of VBA:

```
Debug.Print thisWord & " ";
```

and that line adds a space after it, then the semicolon at the end means don't go down to a newline yet.

Having dropped off the end of the actual text-based number, we've got one too many words, so we take one off and jump out of the For-Next loop (OK, some say it's bad programming technique; I say it works well!):

```
numWords = i - 1
```

```
      Exit For
```

The Immediate mode window of VBA is opened from the View tab, or with Ctrl-G (see the screenshot below).

Find code numbers for each word

```
Dim n(8) As Integer
For i = 1 To numWords
  wdPos = InStr(allNumberWords, ":" & wd(i) & ":")
  leftWords = Left(allNumberWords, wdPos)
  n(i) = Len(leftWords) - Len(Replace(leftWords, ":", ""))
  Debug.Print n(i), wd(i)
Next
```

So we now make up an array of all the numWords numbers equivalent to the words.

First find the position where the word (with colons either end) appears in allNumberWords:

```
      wdPos = InStr(allNumberWords, ":" & wd(i) & ":")
```

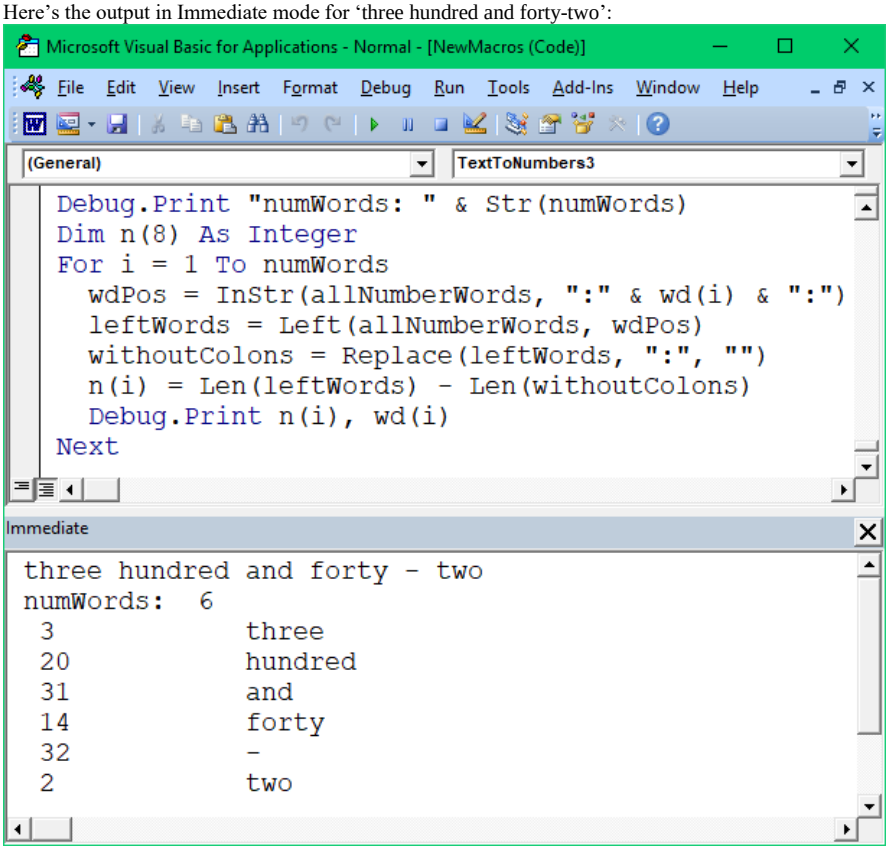
Then pick up leftWords, the words upto but not including the first colon around our word:

```
      leftWords = Left(allNumberWords, wdPos)
```

then we work out how many colons there are getting the length of leftWords then subtracting the length of the string after replacing all the colons with nothing.

```
      n(i) = Len(leftWords) - Len(Replace(leftWords, ":", ""))
```

Here's the output in Immediate mode for 'three hundred and forty-two':



Calculate the number of the text-based number

This has to be done by thinking of all the permutations and combinations of our list of number-words. And it has to be done by thinking carefully about what the different possibilities are for each of the numbers of words. Here are the possibles from the sample above (ignoring the pink ones as I went on to add those once I'd got the basics going), just giving one example of each logically different options:

One word: three, forty, sixteen
Two words: a hundred, one hundred
Three words: seventy-nine
Four words: two hundred and sixteen, two hundred and forty
Six words: three hundred and forty-two

There are five possible values of numWords, so we set up a Select Case of this form:

```
Select Case numWords
  Case 1

  Case 2

  Case 3

  Case 4

  Case 5

  Case 6

  Case Else

End Select
```

Then each of the calculations goes in the gap under each Case number:

```
Select Case numWords
  Case 1
    myResult = n(1)
    If n(1) > 10 Then myResult = 10 * (n(1) - 10)
    If n(1) > 20 Then myResult = n(1) - 10
  Case 2
    If n(2) <> 20 Then ' "hundred"
      Beep
      Exit Sub
    End If
    myResult = n(1) * 100
  Case 3
    If n(2) <> 32 Then ' hyphen
      Beep
      Exit Sub
    End If
    myResult = 10 * (n(1) - 10) + n(3)
  Case 4
    If n(2) <> 20 Then ' "hundred"
      Beep
      Exit Sub
    End If
    myResult = n(4)
    If n(4) > 10 Then myResult = 10 * (n(4) - 10)
    If n(4) > 20 Then myResult = n(4) - 10
    myResult = myResult + 100 * n(1)
```

```
  Case 5
    If n(2) <> 20 Then ' "hundred"
      Beep
      Exit Sub
    End If
    myResult = 100 * n(1) + 10 * (n(3) - 10) + n(5)
  Case 6
    If n(2) <> 20 Then ' "hundred"
      Beep
      Exit Sub
    End If
    myResult = 100 * n(1) + 10 * (n(4) - 10) + n(6)
  Case Else
      Beep
      Exit Sub
End Select
Debug.Print myResult
MsgBox myResult
```

I won't go through them all, but for Case 1 we have:

```
      myResult = n(1)
      If n(1) > 10 Then myResult = 10 * (n(1) - 10)
      If n(1) > 20 Then myResult = n(1) - 10
```

This works by first assuming it's a one to nine number: i.e. the number is just n(1). But then we say, yes but if the number is more than 10, it's in the ten to ninety group. And then again if it's >20 we say, no, it's in the eleven to nineteen, and do the calculation you worked out for homework!

Hopefully, that's enough for you to see basically how they work. Your next homework then is to work out how each of the calculations works.

Finally, I print the result in the Immediate mode and also in a MsgBox, so I can check that it works OK.

Type the number into the text

Now we need to type the number into the text, in place of the text-based number. This is done by the bits that are highlighted.

```
For i = 1 To 8
  thisWord = Trim(rng.Words(i))
  If InStr(allNumberWords, ":" & thisWord & ":") > 0 Then
    wd(i) = thisWord
    Debug.Print thisWord & " ";
    allWords = allWords & thisWord & " "
  Else
    numWords = i - 1
    Exit For
  End If
Next i

rng.MoveEnd wdWord, numWords - 9
rng.MoveEndWhile cset:=" ", Count:=wdBackward
rng.Select

Debug.Print
If wd(1) = "a" Then wd(1) = "one"
```

```
Dim n(8) As Integer
For i = 1 To numWords
  wdPos = InStr(allNumberWords, ":" & wd(i) & ":")
  leftWords = Left(allNumberWords, wdPos)
  n(i) = Len(leftWords) - Len(Replace(leftWords, ":", ""))
  Debug.Print n(i), wd(i)
Next

a = allWords
Select Case numWords
  Case 1
    myResult = n(1)
    If n(1) > 10 Then myResult = 10 * (n(1) - 10)
    If n(1) > 20 Then myResult = n(1) - 10
  Case 2
    If n(2) <> 20 Then ' "hundred"
      Beep
      Exit Sub
    End If
    myResult = n(1) * 100
  Case 3
    If n(2) <> 32 Then ' hyphen
      Beep
      Exit Sub
    End If
    myResult = 10 * (n(1) - 10) + n(3)
  Case 4
    If n(2) <> 20 Then ' "hundred"
      Beep
      Exit Sub
    End If
    myResult = n(4)
    If n(4) > 10 Then myResult = 10 * (n(4) - 10)
    If n(4) > 20 Then myResult = n(4) - 10
    myResult = myResult + 100 * n(1)
  Case 5
    If n(2) <> 20 Then ' "hundred"
      Beep
      Exit Sub
    End If
    myResult = 100 * n(1) + 10 * (n(3) - 10) + n(5)
  Case 6
    If n(2) <> 20 Then ' "hundred"
      Beep
      Exit Sub
    End If
    myResult = 100 * n(1) + 10 * (n(4) - 10) + n(6)
  Case Else
      Beep
      Exit Sub
End Select
Debug.Print myResult
rng.Delete
Selection.TypeText Text:=Trim(Str(myResult))
End Sub
```

In this code:

```
rng.MoveEnd wdWord, numWords - 9
```

```
rng.MoveEndWhile cset:=" ", Count:=wdBackward
```

```
rng.Select
```

I can't quite remember how/why the end of the range, `rng`, had to be moved forwards by `numWords - 9` words (i.e. backwards since numWords is less than 9), but I used the `rng.Select` for debugging, to make the range visible, to make sure I'd got just the right words. (This `numWords - 9` might even be an error, but I changed the method later, so we won't worry about it.)

I also used

```
rng.MoveEndWhile cset:=" ", Count:=wdBackward
```

to bring the selection (well, the range) back past the space.

Then we type the number in place of the range:

```
rng.Delete
Selection.TypeText Text:=Trim(Str(myResult))
```

Ah, but this only works of the rng is actually **selected** by `rng.Select`, because we're using `Selection` to do the typing, i.e. type at the current cursor position.

Make it more user friendly

So far, this only works if the cursor is in the first word of the text-based number, but what we want is to allow the user to *either* put the cursor somewhere (anywhere) in the number *or* put the cursor anywhere to the left of the number. In that way, you can convert one number, then run the macro a second time without bothering to move the cursor.

So we check the word at the cursor and, if it's not a valid number-word then we extend the range right until we do find a number-word – this then is the first word of the text-based number, and out conversion can proceed.

If, However, the first word ***i*** a number-word, we extend the range to the left until we overshoot and find a word that's not a number-word. Then extend the right-hand end of the range to the end of the text-based number.

```
Do
  rng.MoveEnd wdWord, 1
  Debug.Print rng.Text
  thisWord = Trim(rng.Words(rng.Words.Count))
  If InStr("and-", thisWord) > 0 Then thisWord = "x"
Loop Until InStr(allNumberWords, ":" & thisWord & ":") > 0

rng.Collapse wdCollapseEnd
rng.MoveEnd wdWord, -1
Debug.Print rng.Text

gotStart = False
Do While gotStart = False
  rng.MoveStart wdWord, -1
```

```
  Debug.Print rng.Text
  If InStr(allNumberWords, ":" & Trim(rng.Words(1)) & ":") = 0 Then
    gotStart = True
    rng.MoveStart wdWord, 1
    gotStart = True
  End If
  DoEvents
Loop

gotEnd = False
Do While gotEnd = False
  rng.MoveEnd wdWord, 1
  lastWord = Trim(rng.Words(rng.Words.Count))
  If InStr(allNumberWords, ":" & lastWord & ":") = 0 Then
    gotEnd = True
    rng.MoveEnd wdWord, -1
  End If
  Debug.Print rng.Text
  DoEvents
Loop
If Trim(rng.Words(1)) = "and" Then rng.MoveStart wdWord, 1
allWords = rng.Text
numWords = rng.Words.Count
```

So that first block (yellow) is:
Extend the range right by one word.

```
      rng.MoveEnd wdWord, 1
```

Show the range in Imm. mode,

```
      Debug.Print rng.Text
```

Pick up the final word of the range,

```
      thisWord = Trim(rng.Words(rng.Words.Count))
```

To expand on that, I could have done it by using:

```
      numWords = rng.Words.Count
      rng.Words(numWords)
```

But instead, I did it as a single line

If this word is ‘a’, ‘an’, ‘and’ or ‘-’ then it might be just part of the ordinary text, so change it to ‘x’, so that we ignore it and check the following word.

```
      If InStr("and-", thisWord) > 0 Then thisWord = "x"
```

Keep going around this loop until we find a word that ***is*** in our list of number-words, allNumberWords.

```
 Loop Until InStr(allNumberWords, ":" & thisWord & ":") > 0
```

The unhighlighted section reduces the range to just the single number-word:

```
      rng.Collapse wdCollapseEnd
      rng.MoveEnd wdWord, -1
```

That next block (turquoise) is where we find the start of the text-based number (in case there was an ‘a’ for ‘a hundred’):

```
gotStart = False              We haven't found the start yet.
Do While gotStart = False     As long as we're not there yet...
  rng.MoveStart wdWord, -1    ...extend the range left by one word
```

```
gotStart = False
Do While gotStart = False
```

If we haven’t found the start yet, extend the range one word left.

```
  rng.MoveStart wdWord, -1
  Debug.Print rng.Text
```
but if we find a word that's not a number-word...
```
  If InStr(allNumberWords, ":" & Trim(rng.Words(1)) & ":") = 0 Then
```
we **have** found the start (in fact we've gone too far left), so...
```
    gotStart = True
```
move the start of the range one word to the right
```
    rng.MoveStart wdWord, 1
    gotStart = True
  End If
  DoEvents
Loop
```

The grey block extends the range, word by word, to the right, until it goes too far and includes a non-number-word, and then pulls back by one word.

A word of warning about Do Loops

Especially when you're developing a macro, you do have to be careful with Do Loops. If you can the condition for ending the loop wrong, it will go into an infinite. Theoretically, you should be able to click Reset on the VBA toolbar. Unfortunately, Do Loops seem to be very 'tight' such that it's sometimes impossible to escape, and... you did remember to save the macros before you did this trial run, didn't you?

In my case, when I was developing this macro, no, I didn't save it; VBA crashed, I had to restart it, and I lost the prgramming I had done! All I needed to do was press Ctrl-S within VBA before running the macro, and all would have been well.

The other precaution you can take is to add a DoEvents command into the loop. This means that every time through the loop, VBA 'puts it head up' to see if anything is happening; this means it should notice that you've clicked Reset (or clicked Ctrl-Break, if the cursor was in Word when you ran the macro).

A further warning about Do Loops

If, like me, you use loads of keystrokes to run your macros, it's all too easy to initiate a macro when you didn't intend to. So, suppose you launched this macro in a text where there weren't ***any*** number words; it will keep looking, and at best you'll have a long delay, but at worst Word will crash. For that reason, I decided to change the first Do Loop into a For Next loop:

```
gottaWord = False
For i = 1 To 50
  rng.MoveEnd wdWord, 1
  Debug.Print rng.Text
  thisWord = Trim(rng.Words(rng.Words.Count))
  If InStr("aand-", thisWord) = 0 And InStr(allNumberWords, _
       ":" & thisWord & ":") > 0 Then
    If Right(Trim(rng.Text), 6) = "no-one" Then
      gottaWord = False
    Else
      gottaWord = True
      Exit For
    End If
  End If
Next i
rng.Collapse wdCollapseEnd
rng.MoveEnd wdWord, -1
If gottaWord = False Then
  rng.Select
  Beep
  myTime = Timer
  Do
```

```
  Loop Until Timer > myTime + 0.2
  Beep
  Exit Sub
End If
```

This is essentially the same as the yellow block two pages back, but it uses a For Next loop (currently set to 50), so it doesn't go on searching for ever if it can't find a number-word. Instead, if it falls off the end of the loop, it does a double-Beep (yellow block). If it does find a number-word, it exits the For-Next loop (turquoise).

The other item I've added (green) was because, while preparing the video, the word 'no-one' came in the text in between one text-based number and the next, and it converted it to 216 or some such!

I haven't explained some of the jiggery-pockery I used in the calculation section of the different numbers of words in the text-based number. They relate to the US versions of text-based numbers, e.g. 'three hundred forty-two' (five words) and 'four hundred two' (three words).

I also allowed for a missing hyphen, e.g. 'forty two' (two words). But for all other 'odd' situations, I just gave a beep and exited the macro.

Your homework is to work through the calculation code and see which bit does what:

```
Select Case numWords
  Case 1
    myResult = n(1)
    If n(1) > 10 Then myResult = 10 * (n(1) - 10)
    If n(1) > 20 Then myResult = n(1) - 10
  Case 2
    If n(2) = 20 Then ' "hundred"
      myResult = n(1) * 100
    Else
      myResult = 10 * (n(1) - 10) + n(2)
      If myResult < 21 Then
        Beep
        rng.Select
        Exit Sub
      End If
    End If
  Case 3
    myResult = 10 * (n(1) - 10) + n(3)
    If n(2) <> 32 Then ' hyphen
      If n(2) = 20 Then
        myResult = n(3) + 100 * n(1)
      Else
        Beep
        rng.Select
        Exit Sub
      End If
    End If
  Case 4
    If n(2) <> 20 Then ' "hundred"
      Beep
      rng.Select
      Exit Sub
    End If
    myResult = n(4)
    If n(4) > 10 Then myResult = 10 * (n(4) - 10)
    If n(4) > 20 Then myResult = n(4) - 10
    myResult = myResult + 100 * n(1)
  Case 5
```

```
    If n(2) <> 20 Then ' "hundred"
      Beep
      rng.Select
      Exit Sub
    End If
    myResult = 100 * n(1) + 10 * (n(3) - 10) + n(5)
  Case 6
    If n(2) <> 20 Then ' "hundred"
      Beep
      rng.Select
      Exit Sub
    End If
    myResult = 100 * n(1) + 10 * (n(4) - 10) + n(6)
  Case Else
      Beep
      rng.Select
      Exit Sub
End Select
```

That’s it for now. I hope some of this was useful!