VBA (Visual Basic For Applications) Programming Part II

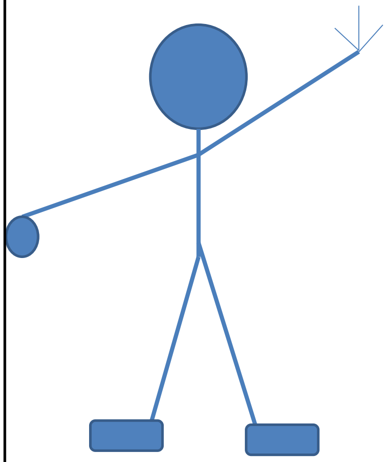
- Objects
- Named constants
- Collections
- Nesting
- Useful VBA functions
- Linking Office applications

Real-World Objects

- You are of course familiar with objects in the everyday world.
  - These are physical entities

- Each object is described by its **properties** (information)
- Each object can have a set of **operations** associated with it (actions)

Example: A Person



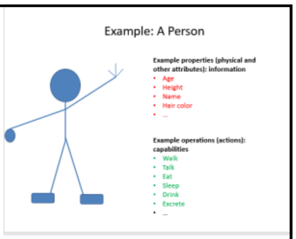
Example properties (physical and other attributes): information

- Age
- Height
- Name
- Hair color
- …

Example operations (actions, in VBA they are called 'methods'): capabilities

- Walk
- Talk
- Eat
- Sleep
- Drink
- Excrete
- …

VBA Object


Example: A Person

Example properties (physical and other attributes): information

- Age
- Height
- Name
- Hair color
- …

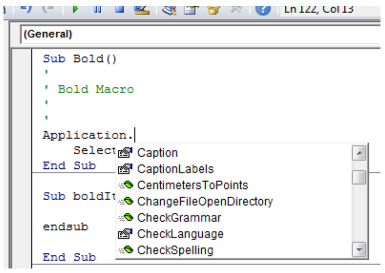
Example operations (actions): capabilities

- Walk
- Talk
- Eat
- Sleep
- Drink
- Excrete
- …

- Similar to everyday objects VBA-Objects have **properties** and **actions**
  - **Properties**: **information** that describe the object
    - E.g., the **name** of a document, **size** of the document, **date modified, number of words** etc.
  - **Capabilities**: **actions** that can be performed (sometimes referred to as 'methods' or 'functions')
    - E.g., **save**, **print**, **spell check** etc.

Common **VBA Objects**

- **`Application`**: the MS-Office program running (for the programs you see in CPSC 203 it will always be MS-Word)
- **`ActiveDocument`**
- **`Selection`**
- When enter one of these keywords in the editor followed by the 'dot' you can see more information.



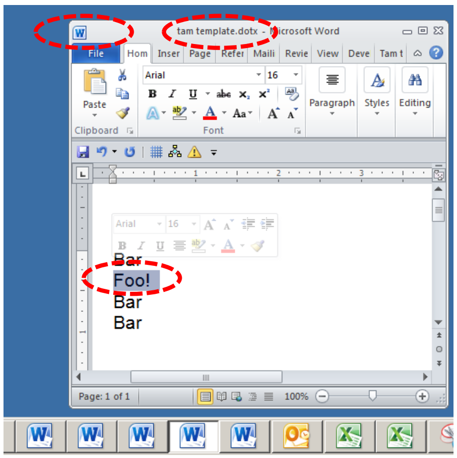
```
Sub Bold()
'
' Bold Macro
'
'
Application.
    Select
End Sub

Sub boldIt

endsub

End Sub
```

```
Caption
CaptionLabels
CentimetersToPoints
ChangeFileOpenDirectory
CheckGrammar
CheckLanguage
CheckSpelling
```

Take advantage of the benefits of VBA:

1. The list of properties and methods is a useful reminder if you can't remember the name
2. If you don't see the pull down then this is clue that you entered the wrong name for the object

Example: What Are The **Three Objects**



- Application:
  - MS-Word
- Active/current Document:
  - "tam template"
- Selection
  - "Foo!"

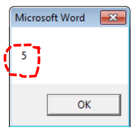
Using Pre-Built Capabilities/Properties Of Objects

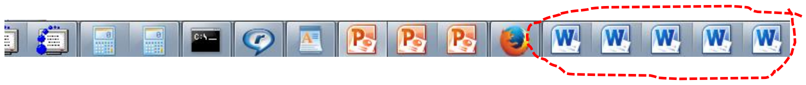
- **Format**:

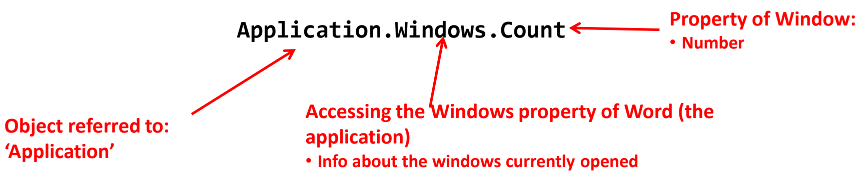
  <Object name>.<method or attribute name>

- **Example**:

```
Sub ApplicationTest()
    MsgBox (Application.Windows.Count)
End Sub
```

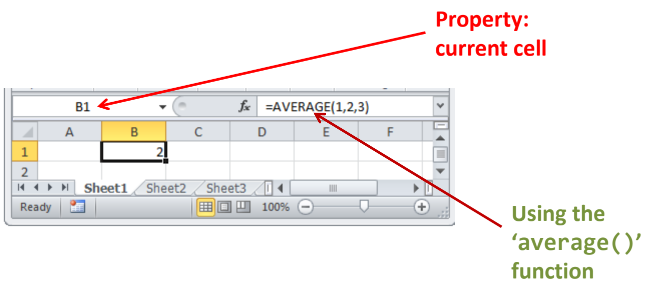




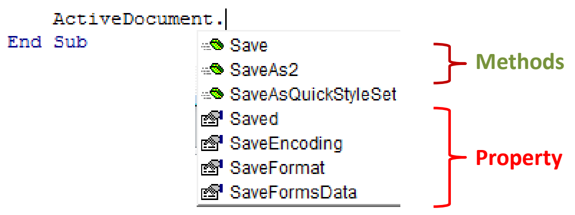

Application.Windows.Count
Property of Window:
• Number
Object referred to: 'Application'
Accessing the Windows property of Word (the application)
• Info about the windows currently opened

Properties Vs. Methods/Functions

- Recall
  - **Property**: information about an object
  - **Method**: capabilities of an object (possible actions)



Properties Vs. Methods: Appearance


```
    ActiveDocument.
End Sub
```

- Similar to functions in MS-Excel some object's methods may require an argument or arguments
- Examples
  - `ActiveDocument.CountNumberedItems`
  - `ActiveDocument.Save`

    No argument required

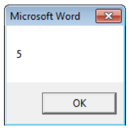
  - `ActiveDocument.SaveAs2("<name>")`

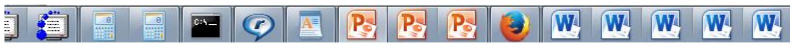
    Argument: New name of document needed

The Application Object

- As mentioned this object is the VBA application running e.g. MS-Word
- **Program illustrating an example usage**: `1applicationObject.docm`

```
Sub ApplicationTest()
    MsgBox (Application.Windows.Count)
End Sub
```





`Application.Windows.Count`

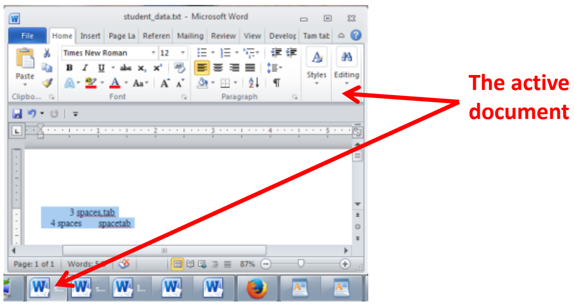
Property of Window:
- Number

Object referred to: 'Application'

Accessing the Windows property of Word (the application)
- Info about the windows currently opened

The ActiveDocument Object

- Quick recap: although you may have many documents open, the 'active document' is the document that you are currently working with:



  - Because it may be easy to confuse documents it's best to only have a single Word document open when writing a VBA program.

Some Attributes Of The ActiveDocument Object

- **Application:** the application/program associated with the document (useful if a VBA macro is linking several applications):details on next slide
- **Content**: the data (text) of the currently active document (needed if you want to perform a text search 'Find' in a VBA program):details later in these notes
- **Name**: the (file) name of the current document (useful for determining the active document if multiple documents are currently open): next slide
- **Path**: the save location of the active document e.g. C:\Temp\ :details on next slide
- **FullName**: the name and save location of the current document :details on next slide
- **HasPassword**: true/false that document is password protected: details on next slide
- **Selection**: the currently select text in the active document (may be empty) :details later in these notes
- **SpellingChecked**: true/false document has been spell checked since document was last edited: :next slide
- **SpellingErrors.Count**: the number of typographical errors

Note: Information for these attributes/properties can be viewed by passing the information as a parameter to a message box
Format: `MsgBox (ActiveDocument.<Attribute Name>)`
Example: `MsgBox (ActiveDocument.SpellingErrors.Count)`

Example Of Accessing **Attributes/Properties**

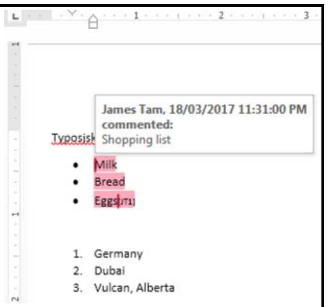
- **Program illustrating an example usage**: `2activeDocumentAttributes.docm`

```
Sub activeDocumentAttributes()
    MsgBox (ActiveDocument.Application)
    MsgBox (ActiveDocument.Name)
    MsgBox (ActiveDocument.Path)
    MsgBox (ActiveDocument.FullName)
    MsgBox ("Spell checked? " & _
      ActiveDocument.SpellingChecked)
    MsgBox ("Password protected? " & _
      ActiveDocument.HasPassword)
    MsgBox ("# typos=" & ActiveDocument.SpellingErrors.Count)
End Sub
```

Some **Methods** Of The `ActiveDocument` Object

- **`Checkspelling`**: exactly as it sounds: next slide
- **`Close`**: closes the active document (different options available)
- **`CountNumberedItems`**: number of bulleted and numbered elements: next slide
- **`DeleteAllComments`**: removes comments from the current document: next slide
- **`Printout`**: prints current active document on the default printer : next slide
- **`Save`**: saves the current document under the same name: next slide
- **`SaveAs2`**: saves the current document under a different name: : next slide
- **`Select`**: select some text in the active document
- **`SendMail()`**: sends an email using MS-Outlook, the currently active document becomes a file attachment

Example Of Using **Methods**


James Tam, 18/03/2017 11:31:00 PM commented:
Shopping list

Typosisk

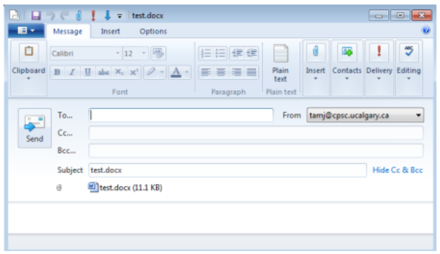
- Milk
- Bread
- Eggs[JT1]

1. Germany
2. Dubai
3. Vulcan, Alberta

- **Program illustrating an example usage**: `3activeDocumentMethods.docm`

```
Sub activeDocumentAttributes()
    ActiveDocument.CheckSpelling
    MsgBox (ActiveDocument.CountNumberedItems)
    ActiveDocument.DeleteAllComments
    ActiveDocument.PrintOut
    ActiveDocument.Save
    ActiveDocument.SaveAs2 ("Copy")
End Sub
```

`ActiveDocument.SendMail()`



To...
Cc...
Bcc...
From tamj@cpsc.ucalgary.ca
Subject test.docx
test.docx (11.1 KB)

- Runs the default email program
- The active document automatically becomes an attachment
- Subject line = name of document
- (For anything more 'fancy' you should use VBA to create and access an MS-Outlook object)

"Finding" Things In A Document

- It can be done in different ways:
- Example (common) 'Find' is an object that is part of the 'Selection' object in a document.
  - JT's note: although it may appear to be confusing at first it doesn't mean that the find (or find and replace) requires text to be selected.
  - Making 'Find' a part of 'Selection' was merely a design decision on the part of Microsoft.
- Example (alternative is JT's preferred approach) 'Find' is an object that is part of the 'Content' object of the 'ActiveDocument'
  - ActiveDocument.Content.Find
  - More details coming up...

One source of information:
http://msdn.microsoft.com/en-us/library/office/aa211953(v=office.11).aspx

Find: Single Replacement

- **Word document containing the macro**: 4simpleFind.docm

```
 sub simpleFind()
    ActiveDocument.Content.Find.Execute
FindText:="tamj",ReplaceWith:="tam"
end Sub
```

'Reminder: The instruction can be broken into two lines without
'an error by using the "line continuation" as a connector

```
ActiveDocument.Content.Find.Execute
FindText:="tamj", _
       ReplaceWith:="tam"
```

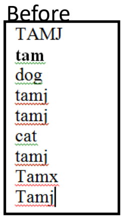
Background for example:
- My old email address (still works): tamj@cpsc.ucalgary.ca
- My new email address: tam@ucalgary.ca
- Incorrect variant: **tamj**@ucalgary.ca

More Complex Find And Replace

- **Word document containing the macro**:
  `findReplaceAllCaseSensitive.docm`

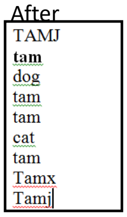
```
Sub findReplaceAllCaseSensitive()
    ActiveDocument.Content.Find.Execute FindText:="tamj", _
      ReplaceWith:="tam", Replace:=wdReplaceAll, _
      MatchCase:=True
End Sub
```

Before



TAMJ
tam
dog
tamj
tamj
cat
tamj
Tamx
Tamj

After



TAMJ
tam
dog
tam
tam
cat
tam
Tamx
Tamj

With, End With

`ActiveDocument.Content.Find`
`.Execute`

- For 'deep' commands that require many levels of 'dots', the '`With`', '`End With`' can be a useful abbreviation.
- **Example**

```
With ActiveDocument.Content.Find
    .Text = "tamj"
```

Anything between the 'find' and 'with' will be preceded in this example with `ActiveDocument.Content.Find:`

Equivalent to (if between the '`with`' and the '`end with`':

```
    ActiveDocument.Content.Find.Text = "tamj"
```

- Previous example, the '`Find`' employing '`With`', '`End With`':
- Also the search and replacement text are specified separately to shorten the '`execute`' (the "`ActiveDocument.Content.Find`" `listed once`)

```
With ActiveDocument.Content.Find
    .Text = "tamj"
    .Replacement.Text = "tam"
    .Execute MatchCase:=True, Replace:=wdReplaceAll
End With
```

'Find text' and 'replacement text' moved here to simplify the '`.execute`'

Find And Replace

- It’s not just limited to looking up text.
- Font effects e.g., bold, italic etc. can also be ‘found’ and changed.

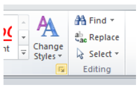
Finding And Replacing Bold Font

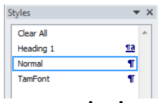
- **Word document containing the macro**: `5findBold.docm`

```
'Removes all bold text
Sub findBold()
   With ActiveDocument.Content.Find
      .Font.Bold = True
      With .Replacement
        .Font.Bold = False
      End With
     .Execute Replace:=wdReplaceAll
   End With
End Sub
```

Finding/Replacing Formatting Styles

- You already have a set of pre-created formatting styles defined in MS-Word.



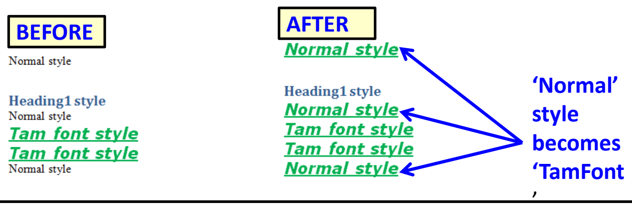


- You can redefine the characteristic of a style if you wish.
- Assume for this example that you wish to retain all existing styles and not change their characteristics.
- But you want to replace all *instances of one style* with another style e.g., all text that is formatted as 'normal' is to become formatted as 'TamFont'
- 'Find' can be used to search (and replace) instances of a formatting style.

Finding/Replacing Formatting Styles (2)

- **Word document containing the macro**: 6findReplaceStyle.docm

```
Sub findReplaceStyle()
   With ActiveDocument.Content.Find
       .Style = "Normal"
       With .Replacement
           .Style = "TamFont"
        End With
       .Execute Replace:=wdReplaceAll
   End With
End Sub
```



Counting The Number Of Occurrences Of A Word

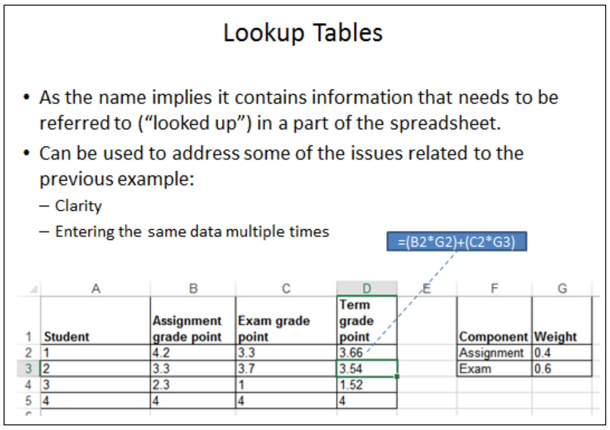
- Example applications:
  - Evaluating resumes by matching skills sought vs. skills listed by the applicant.
  - Ranking the relevance of a paper vs. a search topic by the number of times that the topic is mentioned.
    - Word frequency may be one criteria employed when websites rank search results according to relevance
- Complete Word document containing the macro: `7counting occurences.docm`

Example: Counting Occurrences

```
Sub countingOccurences()
    Dim count As Long
    Dim searchWord As String
    count = 0
    searchWord = InputBox("Word to search for")

    ' Exact match (assignment)
    With ActiveDocument.Content.Find
        Do While .Execute(FindText:=searchWord, Forward:=True, _
          MatchWholeWord:=True) = True
           count = count + 1
        Loop
    End With
    MsgBox ("Exact matches " & count)
End Sub
```

Review: Lookup Tables (For Constants)

- Excel: Lookup tables are used to define values that do not typically change but are referred to in multiple parts of a spreadsheet.



Lookup Tables

- As the name implies it contains information that needs to be referred to ("looked up") in a part of the spreadsheet.
- Can be used to address some of the issues related to the previous example:
  - Clarity
  - Entering the same data multiple times

=(B2*G2)+(C2*G3)

	A	B	C	D	E	F	G
1	Student	Assignment grade point	Exam grade point	Term grade point		Component	Weight
2	1	4.2	3.3	3.66		Assignment	0.4
3	2	3.3	3.7	3.54		Exam	0.6
4	3	2.3	1	1.52			
5	4	4	4	4			

Named Constants

- They are similar to variables: a memory location that's been given a name.
- Unlike variables their contents *cannot* change.
- The naming conventions for choosing variable names generally apply to constants but constants should be all UPPER CASE. (You can separate multiple words with an underscore).
  - This isn't a usual Visual Basic convention but since it's very common with most other languages, you will be required to follow it for this class.
- Example **CONST PI** = 3.14
  - **PI = Named constant**, 3.14 = Unnamed constant
- They are capitalized so the reader of the program can quickly distinguish them from variables.

Declaring Named Constants

- **Format**:

  `Const <Name of constant> = <Expression>`[1]

  JT: it's preceded by the keyword 'const' to indicate that it is a constant/unchanging.

- **Example**:

```
Sub ConstantExample()
   Dim area as Double
   Dim radius as Double
   Const PI = 3.14
   radius = InputBox("Radius")
   area = PI * (radius * radius)
End Sub
```

1 The expression can be any mathematical operation but can't be the result of a function call

Why Use Named Constants

- They can make your programs easier to read and understand
- Example:

  `Income = 315 * 80` No ☹

  Vs.

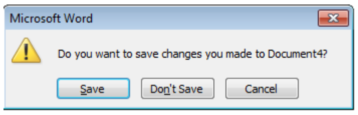
  `Income = WORKING_DAYS_PER_YEAR * DAILY_PAY` Yes ☺

Predefined Constants: **MS-Word Constants**

- Microsoft uses their owning naming convention for predefined named constants.
- Example:
  - `wdPromptToSaveChanges`
- Usage:
  - `ActiveDocument.Close(wdPromptToSaveChanges)`

Closing Documents

- Default action when closing a MS-Word document that has been modified (prompt)


Microsoft Word

Do you want to save changes you made to Document4?

- VBA code to close a document in this fashion:

$$\texttt{ActiveDocument.Close (}\underbrace{\texttt{wdPromptToSaveChanges}}_{\text{Pre-defined constant}}\texttt{)}$$

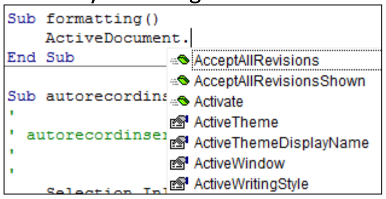
More **Pre-Defined Constants**: Closing Documents

- `ActiveDocument.Close` method
- **Word document containing the macro**: "`8closingActions.docm`"

```
Sub ClosingActions()
    ActiveDocument.Close (<Constant for closing action>)

        'Choose one constant
        wdPromptToSaveChanges
        wdDoNotSaveChanges
        wdSaveChanges

End Sub
```

Formatting A Document

- Entire document:
  - You first need to specify the document or part of a document to be formatted
  - One way is through the '`ActiveDocument`' object



```
Sub formatting()
    ActiveDocument.
End Sub

Sub autorecordin
'
' autorecordinse
'
'
    Selection.In
```

  - Then choose the '`Select`' method of that document.
    - Review: it's a method and not a property because it applies an action: select = selecting the text of the entire document
- Selected text:
  - Only format the currently selected text via the '`Selection`' object).

Formatting Text (Entire Active Document): An Example

- Suppose you want to format a document in the following way
- Entire document
  - Font = Calibri

Formatting: Entire Document

- As mentioned the entire document can be selected.

```
ActiveDocument.Select
```

- Now for the 'selected text' (in this case it's the whole document) access the '`Font`' property and the '`Name`' property of that font and give it the desired name.

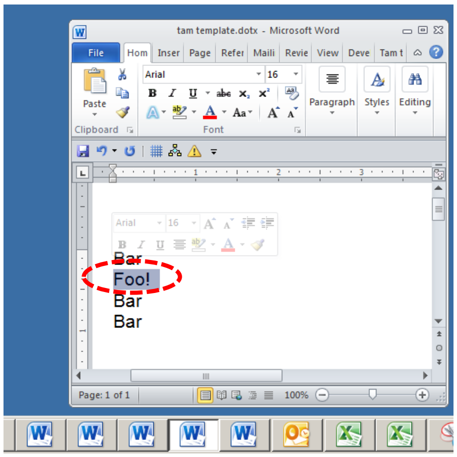
```
Selection.Font.Name = "Calibri"
```

- **Word document containing the macro**: `9formattingEntireDocument`.docm

```
Sub formattingEntireDocument()
    ActiveDocument.Select
    Selection.Font.Name = "Calibri"
End Sub
```

The Selection Object

- This is the currently selected text in a document.
  - It may be empty (nothing selected)



Some **Attributes/Properties** Of The `Selection` Object

- **`Font.Name`**: specify the type (name) of font
- **`Font.Size`**: specify how big is the font
- **`Font.ColorIndex`**: specify the color of the font
- **`Font.UnderLine`**: specify the type of underlining to be applied (or to remove underlining)
- **`Font.Bold`**: allows bolding to change (toggle or set)

Similar to how the Attributes/Properties of `ActiveDocument` Object affect only the currently active document these Attributes/Properties only take effect on the currently selected text (if there's any).

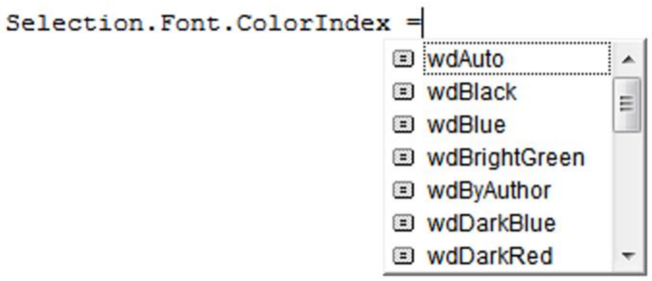
Using The **Selection Object Attributes/Properties**

- **Name of the Word document containing the program**: `10selectionAttributes.docm`

```
Sub selectionObjectAttributes()
    Selection.Font.Name = "Wingdings"  'Must be in quotes
    Selection.Font.Size = 36
    Selection.Font.ColorIndex = wdBlue
    ' Selection.Font.Underline = <Constant for underlining>
    ' wdUnderlineNone, wdUnderlineSingle
    ' e.g. Selection.Font = wdUnderlineSingle

    ' Bolding options
    Selection.Font.Bold = wdToggle ' On/off
    Selection.Font.Bold = True     ' Turn on (false = off)
End Sub
```

Seeing Color (And Under Line Options)

- Use the 'auto complete' feature of VBA to view the options



```
Selection.Font.ColorIndex =
```

Some Methods Of The Selection Object

- ClearFormatting: removes all formatting effects (e.g. bold, italics)
- TypeText: insert the specified text in the VBA program
- Delete: deletes any selected text
- EndKey: move the cursor to the end of the document (covered later in a large example)
- HomeKey: move the cursor to the start of the document (covered later in a large example)
- InsertFile: replace selection with text from the specified file
- (covered in a later example)

Similar to how the method of ActiveDocument Object only affect the currently active document these Attributes/Properties may only take effect on the currently selected text (if there's any).

- Affects selected text: ClearFormatting, Delete
- Does not require text to be selected: TypeText, EndKwy, HomeKey, InsertFile

Using Simple Methods Of The Selection Object

- **Name of the Word document containing the program**: 11selectionMethod.docm
- Try running it with and without some text selected

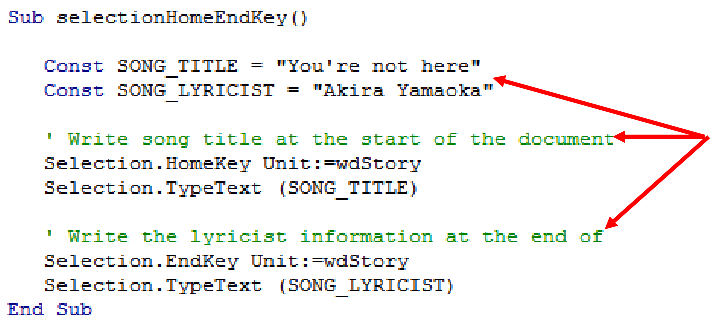
```
Sub selectionObjectMethod()
   Selection.ClearFormatting
   Selection.TypeText ("My new replacement text")
End Sub
```

Writing Text To **Start**/**End**

- **Name of the Word document containing the program**: `12selectionHomeEndKey.docm`
  – HomeKey docs: https://msdn.microsoft.com/en-us/library/office/ff192384.aspx
  – EndKey docs: https://msdn.microsoft.com/en-us/library/office/ff195593.aspx

```
Sub selectionHomeEndKey()
   Const SONG_TITLE = "You're not here"
   Const SONG_LYRICIST = "Akira Yamaoka"
   Selection.HomeKey Unit:=wdStory
   Selection.TypeText (SONG_TITLE)
   Selection.EndKey Unit:=wdStory
   Selection.TypeText (SONG_LYRICIST)
End Sub
```

The Previous VBA Program: Application Of 'Proximity' (Spreadsheet section: C.R.A.*P*.)

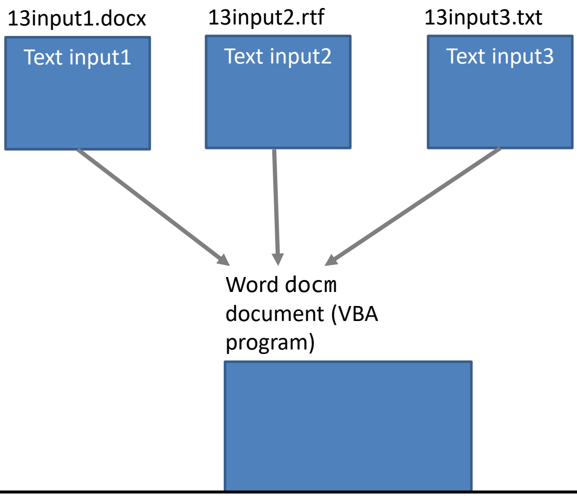


```
Sub selectionHomeEndKey()

   Const SONG_TITLE = "You're not here"
   Const SONG_LYRICIST = "Akira Yamaoka"

   ' Write song title at the start of the document
   Selection.HomeKey Unit:=wdStory
   Selection.TypeText (SONG_TITLE)

   ' Write the lyricist information at the end of
   Selection.EndKey Unit:=wdStory
   Selection.TypeText (SONG_LYRICIST)
End Sub
```

- **Related parts of the program are grouped together**
- **Each part is separated with whitespace**

Inserting Text

- Example files (must all be in the same folder)



Automatically **Inserting Text** Into A Word Document

- **Name of the Word document containing the program**: `13selectionInsertingText.docm`

```
Sub insertingText()
    Selection.InsertFile ("13input1.docx")
    Selection.InsertFile ("13input2.docx")
    Selection.InsertFile ("13input3.docx")
End Sub
```

The Selection Object again

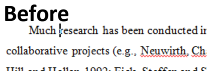
- With a approaches if no text was selected then the program would produce no visible effect.

```
Sub SelectedFontChange()
    Selection.Font.Bold = wdToggle
End
```

- The program could automatically select text for you "expanding" the selection.

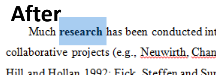
```
Sub AutoSelectedFontChange()
    Selection.Expand
    Selection.Font.Bold = wdToggle
End Sub
```

Before



Much research has been conducted ir
collaborative projects (e.g., Neuwirth, Ch
Hill and Hollan 1992; Fick, Steffen and S

After



Much **research** has been conducted int
collaborative projects (e.g., Neuwirth, Chan
Hill and Hollan 1992; Fick, Steffen and Sur

Constants For The Selection Object

Name of constant	Meaning of constant
wdSelectionIP	No text selected
wdSelectionNormal	Text (e.g., word, sentence) has been selected
wdSelectionShape	A graphical shape (e.g., circle, text book) has been selected

Application of these constants coming up on the next slide

The `Selection` Object And A Practical Application Of Branching

- An example application of branching: check if a selection has been made and only apply the selection if that is the case.
  – Checking if a condition is true
- **Word document containing the macro:**
  “14selectionExample.docm”

```
Sub checkSelection()
    If Selection.Type = wdSelectionIP Then
        MsgBox ("No text selected, nothing to change")
    Else
        Selection.Font.Bold = wdToggle  'wdToggle, constant
    End If
End Sub
```

Applications Of Branching

- **Checking state**

```
  IF(program is in some state) then
      Program reacts
  End
```

- **Example 1:**

```
  If (Application.CapsLock = True) Then
      MsgBox ("Caution: Caps Lock is On!“)
  End If
```

- **Example 2:**

```
  age = InputBox("Age: ")
  If (age < 0) Then
      MsgBox ("Age cannot be negative")
  End If
```

Applications Of Branching

- Recall this example from earlier (tutorial)

```
Selection.Font.Bold = wdToggle
```

- If no text is selected nothing happens but the user is not informed.
- Modification of the program using IF-branches:
  - Checking if text has been selected and if not displaying an error popup message.
- **Example 3**:

```
If (Selection.Type = wdSelectionIP) Then
    MsgBox ("No text selected, nothing to change")
Else
    Selection.Font.Bold = wdToggle 'wdToggle, constant
End If
```

Constants for the Selection object

wdSelectionIP	No text selected
wdSelectionNormal	Text selected

Application Branching: Marking Program (If There Is Time)

- **Word document containing the macro: “**`15Marking program.docm`”
- Synopsis:
  - The program spells checks the document
    - Assume each document includes the name of the person in the file name
  - If the number of errors meets a cut-off value then it’s a ‘fail’
  - Otherwise it’s a pass
  - The feedback is ‘written’ to the beginning of the document using a specific font and several font effects in order to stand out
    - The message is customized with the person’s name at the beginning of the feedback

Marking Program

```
Sub MarkingForSpelling()
    Dim totalTypos As Integer
    Const MAX_TYPOS = 1
    Dim currentDocument As String
    Dim feedback As String

    'Get Name of current document
    currentDocument = ActiveDocument.Name

    'Tally the number of typos
    totalTypos = ActiveDocument.SpellingErrors.Count

    'Feedback is prefaced by student(document) name
    feedback = currentDocument & " marking feedback..."
```

Marking Program (2)

```
    ' HomeKey move to the home position (start of document)
    Selection.HomeKey Unit:=wdStory

    'Recall: before this feedback just = document name and
    'an indication that feedback is coming
    If (totalTypos > MAX_TYPOS) Then
        feedback = feedback & ": Too many typographical errors:
                                 Fail"
    Else
        feedback = feedback & ": Pass"

    End If

    ' Chr(11) adds a newline (enter) to the end of feedback
    feedback = feedback & Chr(11) & Chr(11)

    ' Alternative use the constant vbCr (VB cursor return)
```

Marking Program (3)

```
    ' Font effects to make the feedback stand out
    Selection.Font.ColorIndex = wdRed
    Selection.Font.Size = 16
    Selection.Font.Name = "Times New Roman"

    ' Write feedback into the document
    Selection.TypeText (feedback)

End Sub
```

Collection

- An object that consists of other objects
  - Real World example: a book consists of pages, a library consists of books
- Example: The *`Documents`* collection will allow access to the documents that have been opened.
- Access to a collection rather than the individual objects may be time-saving shortcut.
  - Instead of manually closing all open documents this can be done in one instruction:

```
Documents.close
```

Types Of **Collections**

- Some Attributes/Properties of a document that return a collection.
  - `Documents`: currently open documents
  - `Shapes:` MS-Word shapes in a document (rectangles, circles etc.)
  - `InlineShapes:` images (pictures inserted) into a Word document
  - `Tables`: tables in a document
    - E.g., `ActiveDocument.Tables` –accesses all the tables in your document
    - `ActiveDocument.Tables(1)` –access to the first table in a document.
  - `Windows`: briefly introduced at the start of this section of notes

`Documents` Collection For **Printing: Multiple Documents**

- Printing all the documents currently open in MS-Word.
  - Take care that you don't run this macro if you have many documents open and/or they are very large!
  - **Word document containing the macro example:** "`16printMultipleDocumentst.docm`"

```
Sub PrintDocumentsCollection()
    Dim numDocuments As Integer
    Dim count As Integer
    numDocuments = Documents.count
    count = 1
    Do While (count <= numDocuments)
        Documents.Item(count).PrintOut
        count = count + 1
    Loop
End Sub
```

Learning: another practical application of looping e.g., automatically open multiple documents, make changes, print and save them without user action needed

Accessing Shapes And Images (If There Is Time)

- (VBA specific)
  - `Shapes` (basic shapes that are drawn by Word) ■ ★ ▲
  - `InlineShapes` (images that are created externally and inserted into Word)
- Both collections accessed via the `ActiveDocument` object:
  - `ActiveDocument.Shapes`: access to all the shapes in the currently active Word document
    - `ActiveDocument.Shapes(<`*`index`*`>):` access to shape #*i* in the document
  - `ActiveDocument.InlineShapes`: access to all the images in the currently active Word document
    - `ActiveDocument.InlineShapes(<`*`index`*`>):` access to image #*i* in the document

Example: Accessing Shapes And Images

Word document containing the complete macro:
"`17accessingImagesFigures.docm`"

```
Dim numImages As Integer
Dim numShapes As Integer

numImages = ActiveDocument.InlineShapes.Count
numShapes = ActiveDocument.Shapes.Count

MsgBox ("Images=" & numImages)
MsgBox ("Simple shapes=" & numShapes)
```

Example: Accessing Shapes And Images (2)

```
' Checks expected # images and alters first & third
If (numImages = 4) Then
    ActiveDocument.InlineShapes(1).Height = _
      ActiveDocument.InlineShapes(1).Height * 2
    ActiveDocument.InlineShapes(3).Height = _
      ActiveDocument.InlineShapes(3).Height * 2
End If

' Checks expected # shapes, alters 2nd & 6th
' Deletes the first shape
If (numShapes = 6) Then
    ActiveDocument.Shapes(2).Width = _
      ActiveDocument.Shapes(2).Width * 4
    ActiveDocument.Shapes(6).Fill.ForeColor = vbRed
    ActiveDocument.Shapes(1).Delete
End If
```

Nesting

- Nesting refers to an item that is “inside of” (or “nested in”) some other item.
- Recall from ‘spreadsheets’ nesting refers to an ‘IF-function’ that is inside of another ‘IF-function’
  - Example (assume that the respondent previously indicated that his or her birthplace was an Alberta city)
  - Select the AB city in which you were born
    1. Airdrie
    2. Calgary
    3. Edmonton

       …
    - Selecting Airdrie excludes the possibility of selecting Calgary
    - Cities listed later are ‘nested’ in earlier selections)
- Nesting in programming (VBA) refers to IF-branches and Do-While loops that are inside of each other

Nesting

- Nesting: one structure is contained within another
  - Nested branches:

```
If (Boolean) then
    If (Boolean) then
        ...
    End If
End if
```

- Branches and loops can be nested within each other

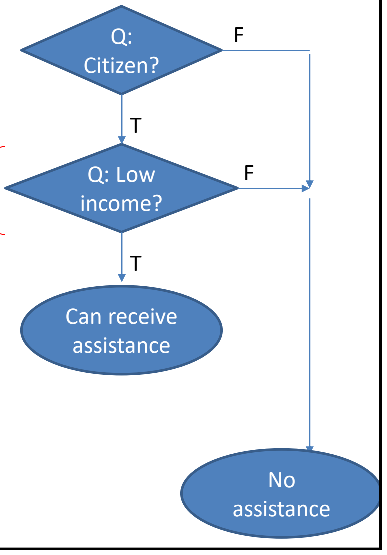
```
Do while (Boolean)                    If (Boolean) then
    If (Boolean) then                     Do while (Boolean)
        ...                                   ...
    End if                                Loop
Loop

Do while (Boolean)
    Do while (Boolean)
        ...
    Loop
Loop
```

Recognizing When Nesting Is Needed

- **Scenario 1**: A second question is asked if a first question answers true:
  - Example: If it's true the applicant is a Canadian citizen, then ask for the person's income (checking if eligible for social assistance).
  - Type of nesting: an IF-branch nested inside another IF-branch

```
If (Boolean) then
    If (Boolean) then
        ...
    End If
End if
```

Nested branch/IF



Example #1: Nested IFs

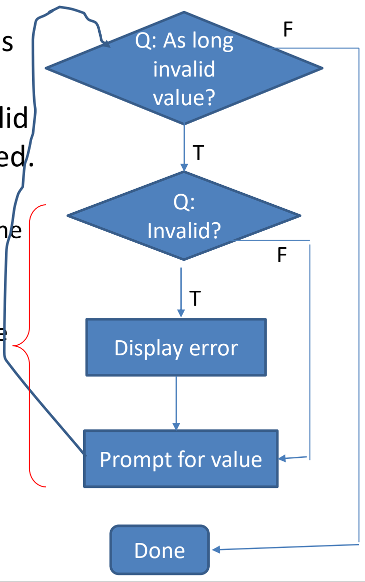
- Word document containing the example: `18nestingIFinsideIF.docm`

```
Sub nestedCase1()
    Dim country As String
    Dim income As Long
    Const INCOME_CUTOFF = 24000
    country = InputBox("What is your country of citizenship?")
    If (country = "Canada") Then
        income = InputBox("What is your income $")
        If (income <= INCOME_CUTOFF) Then
            MsgBox ("Citizenship: " & country & "; " & _
              "Income $" & income & _
              ": eligible for assistance")
        End If
    End If
End Sub
```

Recognizing When **Nesting** Is Needed

- **Scenario 2A**: As long some condition is met a question will be asked. As the question is asked if the answer is invalid then an error message will be displayed.
  - Example: While the user entered an invalid value for age (too high or too low) then if the age is too low an error message will be displayed.
  - Type of nesting: an IF-branch nested inside of a `Do-While` loop

```
 Do While (Boolean)
     If (Boolean) then
         ...
     End If
 Loop
```



Example 2A: **IF Nested** Inside A Do-While

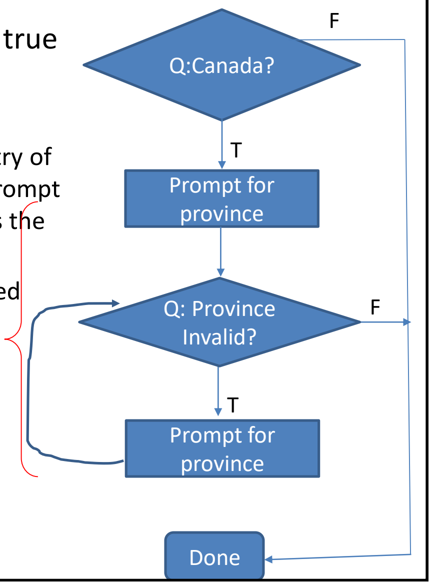
- Word document containing the example: `19nestingIFinsideWHILE.docm`

```
Sub nestedCase2A()
    Dim age As Long
    Const MIN_AGE = 1
    Const MAX_AGE = 118
    age = InputBox("How old are you (1-118)?")
    Do While ((age < MIN_AGE) Or (age > MAX_AGE))
        If (age < MIN_AGE) Then
            MsgBox ("Age cannot be lower than " & _
              MIN_AGE & " years")
        End If
        age = InputBox("How old are you (1-118)?")
    Loop
    MsgBox ("Age=" & age & " is age-okay")
End Sub
```

Recognizing When **Nesting** Is Needed

- **Scenario 2B**: If a question answers true then check if a process should be repeated.
  - Example: If the user specified the country of residence as Canada then repeatedly prompt for the province of residence as long as the province is not valid.
  - Type of nesting: a Do-While loop nested inside of an IF-branch

```
If (Boolean) then
    Do While (Boolean)
        ...
    Loop
End If
```



Example 2B: **Do-While Nested** Inside An IF

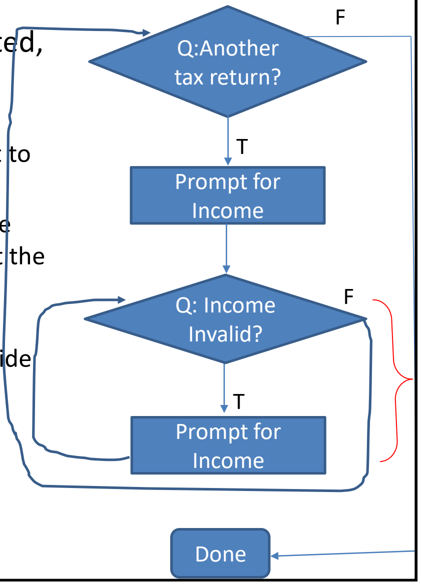
- Word document containing the example: `20nestingWHILEinsideIF.docm`

```
Dim country As String
Dim province As String
country = InputBox("What is your country of citizenship?")
If (country = "Canada") Then
    province = InputBox("What is your province of " & _
       "citizenship?")
    Do While ((province <> "AB") And (province <> "BC"))
        MsgBox ("Valid provinces: AB, BC")
        province = InputBox("What is your province of" & _
          " citizenship?")
    Loop
End If
MsgBox ("Country: " & country & ", " & "Province: " & _
  " province)
```

Recognizing When **Nesting** Is Needed

- **Scenario 3**: While one process is repeated, repeat another process.
  – More specifically: for each step in the first process repeat the second process from start to end
  – Example: While the user indicates that he/she wants to calculate another tax return prompt the user for income, while the income is invalid repeatedly prompt for income.
  – Type of nesting: a `Do-While` loop nested inside of an another `Do-While` loop

```
Do While (Boolean)
    Do While (Boolean)
        ...
    Loop
Loop
```



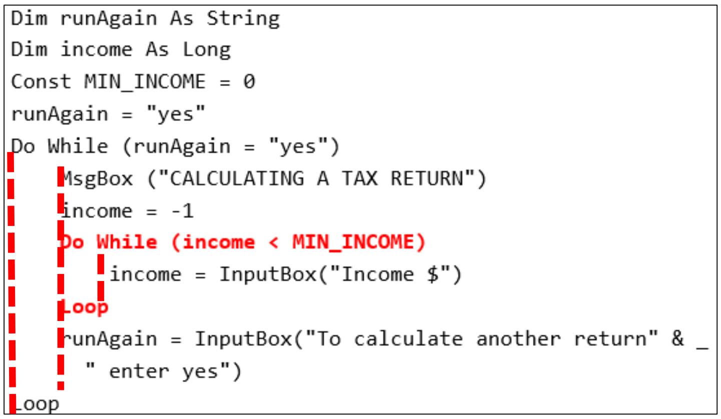
Example #3: **Do-While Nested** Inside Another Do-While

- Word document containing the example: `21nestingWHILEinsideWHILE.docm`

```
Dim runAgain As String
Dim income As Long
Const MIN_INCOME = 0
runAgain = "yes"
Do While (runAgain = "yes")
    MsgBox ("CALCULATING A TAX RETURN")
    income = -1
    Do While (income < MIN_INCOME)
        income = InputBox("Income $")
    Loop
    runAgain = InputBox("To calculate another return" & _
      " enter yes")
Loop
```
a

Application Of Alignment (C.R.A.P.): Nesting

- Allows program structure (nesting levels) to be easily determined:
  - What statements execute as which body



```
Dim runAgain As String
Dim income As Long
Const MIN_INCOME = 0
runAgain = "yes"
Do While (runAgain = "yes")
    MsgBox ("CALCULATING A TAX RETURN")
    income = -1
    Do While (income < MIN_INCOME)
        income = InputBox("Income $")
    Loop
    runAgain = InputBox("To calculate another return" & _
      " enter yes")
Loop
```

Example: Nesting

1. Write a program that will count out all the numbers from one to six.
2. For each of the numbers in this sequence the program will determine if the current count (1 – 6) is odd or even.
   a) The program display the value of the current count as well an indication whether it is odd or even.

- Which Step (#1 or #2) should be completed first?

Step #1 Completed: Now What?

- For each number in the sequence determine if it is odd or even.
- This can be done with the modulo (remainder) operator: MOD
  - An even number modulo 2 equals zero (2, 4, 6 etc. even divide into 2 and yield a remainder or modulo of zero).
  - `If (counter MOD 2 = 0)` then `'Even`
  - An odd number modulo 2 does not equal zero (1, 3, 5, etc.)
- Pseudo code visualization of the problem

```
Loop to count from 1 to 6
    Determine if number is odd/even and display message
End Loop
```

  - Determining whether a number is odd/even is a part of counting through the sequence from 1 – 6, checking odd/even is nested within the loop

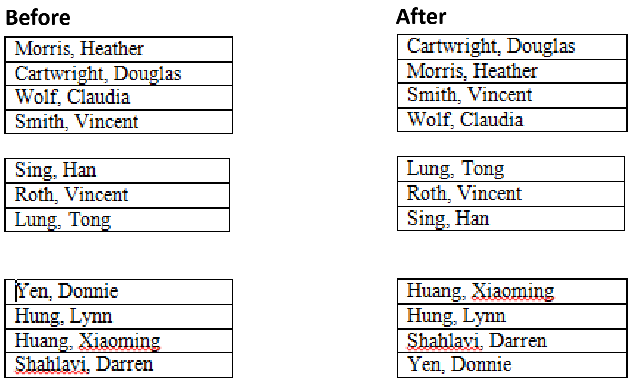
Accessing Tables (If There Is Time)

- The tables in the currently active Word document can be made through the `ActiveDocument` object:
  - `ActiveDocument.Tables:` accesses the '`tables`' collection (all the tables in the document).
  - `ActiveDocument.Tables(<integer 'i'>):` accesses table # *i* in the document
  - `ActiveDocument.Tables(1).Sort`: sorts the first table in the document (default is ascending order)

Simple Example: Sorting Three Tables

- Instructions needed for sorting 3 tables

```
ActiveDocument.Tables(1).Sort
ActiveDocument.Tables(2).Sort
ActiveDocument.Tables(3).Sort
```



Before

Morris, Heather
Cartwright, Douglas
Wolf, Claudia
Smith, Vincent

Sing, Han
Roth, Vincent
Lung, Tong

Yen, Donnie
Hung, Lynn
Huang, Xiaoming
Shahlavi, Darren

After

Cartwright, Douglas
Morris, Heather
Smith, Vincent
Wolf, Claudia

Lung, Tong
Roth, Vincent
Sing, Han

Huang, Xiaoming
Hung, Lynn
Shahlavi, Darren
Yen, Donnie

Previous Example

- Critique of the previous approach: the program ‘worked’ for the one document with3 tables but:
  - What if there were more tables (cut and paste of the sort instruction is wasteful)?
  - What if the number of tables can change (i.e., user edits the document)
- Notice: The process of sorting just repeats the same action but on a different table.

```
ActiveDocument.Tables(1).Sort
ActiveDocument.Tables(2).Sort
ActiveDocument.Tables(3).Sort
```

- Looping/repetition can be applied reduce the duplicated statements

Revised Example: Sorting Tables With A Loop

Word document containing the complete macro: “22sortingTables.docm”

```
Dim CurrentTable As Integer
Dim NumTables As Integer
NumTables = ActiveDocument.Tables.Count
If NumTables = 0 Then
    MsgBox ("No tables to sort")
Else
    CurrentTable = 1
    Do While (CurrentTable <= NumTables)
        MsgBox ("Sorting Table # " & CurrentTable)
        ActiveDocument.Tables(CurrentTable).Sort
        CurrentTable = CurrentTable + 1
    Loop
End If
```

Result: Sorting Tables

- **Before**

A
B
c

Z
B
a

Morris Heather	Heroine
Adama, Lee	CAG
Adama, Bill	Commander

- **After**

A
B
c

a
B
Z

Adama, Bill	Commander
Adama, Lee	CAG
Morris Heather	Heroine

More On Sort

- A **handy parameter** that can be used to configure how it runs.
- **Format**

  `Sort (<Boolean to Exclude header – True or False>)`

- **Example**
  - `ActiveDocument.Tables(CurrentTable).Sort(True)`
  - Before

Name	Title
Tam, James	Boring
Bond, James	Spy

  - After

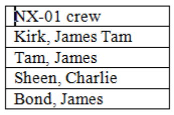
Name	Title
Bond, James	Spy
Tam, James	Boring

Second Sorting Example: **Exclude Headers**

- **Document containing the macro**: "23sortingTablesExcludeHeader.docm"

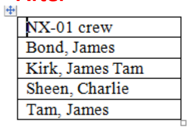
```
Dim CurrentTable As Integer
Dim NumTables As Integer
NumTables = ActiveDocument.Tables.Count
If NumTables = 0 Then
    ' Don't bother sorting
    MsgBox ("No tables to sort")
Else
    CurrentTable = 1
    Do While (CurrentTable <= NumTables)
        MsgBox ("Sorting Table # " & CurrentTable)
        ActiveDocument.Tables(CurrentTable).Sort (True)
        CurrentTable = CurrentTable + 1
    Loop
End If
```

Before



NX-01 crew
Kirk, James Tam
Tam, James
Sheen, Charlie
Bond, James

After



NX-01 crew
Bond, James
Kirk, James Tam
Sheen, Charlie
Tam, James

The DIR Function

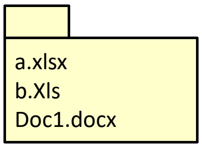
- It can be used to go through all the documents in a folder (this will be illustrated gradually in advanced examples but the first one will be rudimentary and only access single documents)
- It can access sub-folders and sub-sub folders of a folder (very advanced use: well beyond the scope of the this course)
- Basic use: this function takes as inpu a location (e.g., C:\temp\) and a filename as an argument and it determines if the file exists at the specified location.
  - If the file is found at this location then the function returns the name of the file.
  - If the file is not found at this location then the function returns an empty string (zero length)
  - Getting the file name can then allow that document to be opened

Simple Use Of The DIR Function

- **Word document containing the macro example:** `24DIRFunctionSimple.docm`

```
Dim location As String
Dim filename As String
Dim result As String
location = "C:\temp\203\dirExample1\"
result = Dir(location)
MsgBox (result)
filename = "Doc1.docx"
result = Dir(location & filename)
MsgBox (result)
result = Dir(location & "*.xls*")
MsgBox (result)
```

Contents of folder “dirExample1”



Output of the message boxes:
- a.xlsx
- Doc1.docx
- a.xlsx

Example: Using Dir To Check If File Exists (2)

- **Word document containing the macro example:** `25DIRFunctionIntermediate.docm`

```
Sub openExistingDocument()
    Dim filename As String
    Dim checkIfExists As String
    Dim last As Integer

    filename = InputBox ("Enter the path and name of file to
       open e.g., 'C:\temp\tam.docx'")
    ' Error case: nothing to open, user entered no info
    If (filename = "") Then
        MsgBox ("You entered a blank file name")
```

Example: Using `Dir` To Check If File Exists (3)

```
' No error: non-empty info entered
Else
    checkIfExists = Dir(filename)
    If (Len(checkIfExists) = 0) Then
        MsgBox ("File does not exist the specified
           location" & filename)
    Else
        MsgBox ("File exists opening")
        Documents.Open (filename)
   End If
End If
End Sub
```

Practical Use Of `Dir`: Access Each File In A Directory

- **Word document containing the macro example:** `26loopFolder.docm`

```
Dim directoryPath As String
Dim currentFile As String
directoryPath = "C:\temp\203\dirExample2\"
currentFile = Dir(directoryPath)
If (currentFile = "") Then
    MsgBox ("No files in directory: " & directoryPath)
End If
Do While (currentFile <> "")
    MsgBox (currentFile) ' Display file name in popup
    currentFile = Dir  ' Move onto next document in folder
 Loop
```

Alternate Version: Access Only **Word Documents**

- **Word document containing the macro example:** `27loopWordFolder.docm`

```
Sub openAllWordDocumentsInFolder()
    Dim directoryPath As String
    Dim currentFile As String

    directoryPath = "C:\temp\203\dirExample3\"
    currentFile = Dir(directoryPath & "*.doc*")


    If (currentFile = "") Then
        MsgBox ("No Word documents in location " &
          directoryPath)
    End If
    Do While (currentFile <> "")
        MsgBox ("Opening " & currentFile)
        Documents.Open (directoryPath & currentFile)
        currentFile = Dir
    Loop
End Sub
```

Applying Many Of The Previous Concepts In A Practical Example & Linking Documents And (If There's Time)

- As you are aware different programs serve different purposes:
  - Database: storing and retrieving information
  - Spreadsheet: performing calculations, displaying graphical views of results
  - Word processor: creating text documents with many features for formatting and laying out text
- VBA allows the output of one program to become the input of another program.
  - Although this can be done 'manually' (reading the documents and typing in changes) if the dataset is large this can be a tedious and error-prone process
  - VBA can be used to automate the process

Processing All Documents In A Folder: Overview

- Overview of the process (may be very helpful for the assignment)

```
While (There are unprocessed documents in folder)
    Open next unprocessed document
    ' (This opened document becomes the active document)
    Process the active document
    Move onto the next document
```

JT's note: this is one of the more complete examples to use as a guide for your solution to the final feature of A3

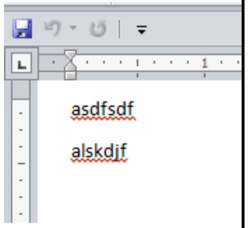
Processing All Documents In A Folder: VBA Program

- **Word document containing the macro**: "`28processAllFolderWordDocuments.docm`"

```
Sub processFolderDocuments()
    Const FOLDER_PATH As String = "C:\temp\203\dirExample3"
    Dim currentFile As String
    Dim wordCount As Long
    currentFile = Dir(FOLDER_PATH & "*.doc*")
    Do While (currentFile <> "")
        Documents.Open (FOLDER_PATH & currentFile)
        wordCount = ActiveDocument.Words.count
        MsgBox (currentFile & "# words in doc " & wordCount)
        currentFile = Dir
    Loop
End Sub
```

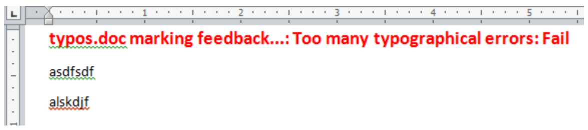
Revised Marking Program (If There Is Time)

- **Word document containing the macro**: "29markAllFolderDocuments.docm"


asdfsdf

alskdjf

```
Sub markAllFolderDocuments()
   Const MAX_TYPOS = 1
   Const LARGER_FONT = 14
   Dim directoryPath As String
   Dim currentFile As String
   Dim totalTypos As Integer
   Dim feedback As String
```


typos.doc marking feedback...: Too many typographical errors: Fail

asdfsdf

alskdjf

Revised Marking Program (2)

```
directoryPath = InputBox("Location and name of folder
  containing assignments (e.g., C:\grades\")
currentFile = Dir(directoryPath & "*.doc*")

If (directoryPath = "") Then
    MsgBox ("No Word documents in specified folder,
             looking in default location C:\Temp\")
    directoryPath = "C:\Temp\"
End If
```

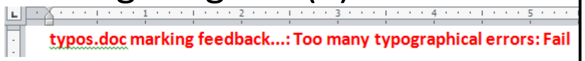
Revised Marking Program (3)

e.g. Feedback for "Typos.docx" = "Typos marking feedback…"

```
Do While (currentFile <> "")
    Documents.Open (directoryPath & currentFile)
    currentDocument = ActiveDocument.Name
    totalTypos = ActiveDocument.SpellingErrors.Count
    feedback = currentDocument & " marking feedback..."
    Selection.HomeKey Unit:=wdStory
    If (totalTypos > MAX_TYPOS) Then
        feedback = feedback & ": Too many typographical
                                errors: Fail"
    Else
        feedback = feedback & ": Pass"
    End If
    feedback = feedback & vbCr
    Selection.Text = feedback
    ' Loop body continued on next page
```

e.g. Feedback for "Typos.docx" = "typos.doc marking feedback…: Too many typographical errors: Fail"

Revised Marking Program (4)


typos.doc marking feedback...: Too many typographical errors: Fail

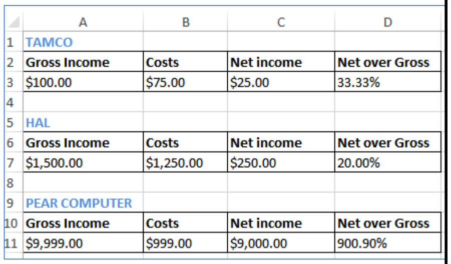
```
    ' Loop body continued from previous page
    With Selection.Font
        .Bold = True
        .Size = LARGER_FONT
        .ColorIndex = wdRed
    End With
    ActiveDocument.Close (wdSaveChanges)
    currentFile = Dir
  Loop
End Sub
```

Example Problem

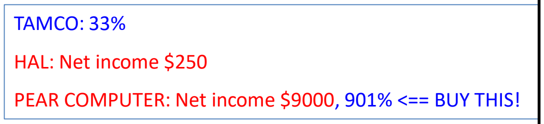
- Financial statements (monetary data) about many companies can be stored in a spreadsheet where an analysis can be performed e.g. does the company have enough $$$ on hand to meet its financial commitments.
- This information can be read into a VBA program which can further evaluate the data.
- The results can be presented in Word using the numerous text formatting features to highlight pertinent financial information.
- **Names of the documents used in this example**:
  - `FNCE.xlsx` (contains the financial data: program input)
  - `30spreadSheetAnalyzer.docm` (contains the VBA program as well as the presentation of results: program output)

Spread Sheet Analyzer

```
Sub spreadsheetAnalyzer()
    Const MIN_INCOME = 250
    Const MIN_RATIO = 25

    Const PERCENT = 100
    Dim company1 As String
    Dim income1 As Long
    Dim ratio1 As Long
    Dim company2 As String
    Dim income2 As Long
    Dim ratio2 As Long
    Dim company3 As String
    Dim income3 As Long
    Dim ratio3 As Long
    Dim comment1 As String
    Dim comment2 As String
    Dim comment3 As String
```



	A	B	C	D
1	TAMCO			
2	Gross Income	Costs	Net income	Net over Gross
3	$100.00	$75.00	$25.00	33.33%
4				
5	HAL			
6	Gross Income	Costs	Net income	Net over Gross
7	$1,500.00	$1,250.00	$250.00	20.00%
8				
9	PEAR COMPUTER			
10	Gross Income	Costs	Net income	Net over Gross
11	$9,999.00	$999.00	$9,000.00	900.90%



TAMCO: 33%

HAL: Net income $250

PEAR COMPUTER: Net income $9000, 901% <== BUY THIS!

Spread Sheet Analyzer (2)

Object =
Type for any MS-Office variable
https://msdn.microsoft.com/

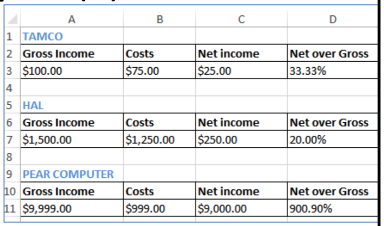
```
Dim excel As Object
Set excel = CreateObject("excel.application")
excel.Visible = True

Dim workbook
Dim location As String
location = InputBox("Path and name of spreadsheet e.g.
                     C:\Temp\FNCE.xlsx")
Set workbook = excel.workbooks.Open(location)
```

Spread Sheet Analyzer (2)

Object =
Type for any MS-Office variable
https://msdn.microsoft.com/

```
Dim excel As Object
Set excel = CreateObject("excel.application")
excel.Visible = True

Dim workbook
Dim location As String
location = InputBox("Path and name of spreadsheet e.g.
                     C:\Temp\FNCE.xlsx")
Set workbook = excel.workbooks.Open(location)
```

Spread Sheet Analyzer (3)

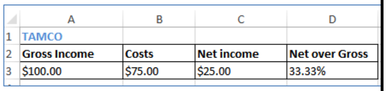


	A	B	C	D
1	TAMCO			
2	Gross Income	Costs	Net income	Net over Gross
3	$100.00	$75.00	$25.00	33.33%
4				
5	HAL			
6	Gross Income	Costs	Net income	Net over Gross
7	$1,500.00	$1,250.00	$250.00	20.00%
8				
9	PEAR COMPUTER			
10	Gross Income	Costs	Net income	Net over Gross
11	$9,999.00	$999.00	$9,000.00	900.90%

```
' Get company names
company1 = excel.Range("A1").Value
company2 = excel.Range("A5").Value
company3 = excel.Range("A9").Value

' Get net income and ratio
income1 = excel.Range("C3").Value
ratio1 = excel.Range("D3").Value * PERCENT
income2 = excel.Range("C7").Value
ratio2 = excel.Range("D7").Value * PERCENT
income3 = excel.Range("C11").Value
ratio3 = excel.Range("D11").Value * PERCENT

' Move the selection to the top of the Word document
Selection.HomeKey Unit:=wdStory
```

TAMCO: 33%

Spread Sheet Analyzer (4): First Company

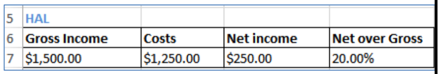


	A	B	C	D
1	TAMCO			
2	Gross Income	Costs	Net income	Net over Gross
3	$100.00	$75.00	$25.00	33.33%

```
comment1 = company1 & ": "
If (income1 >= MIN_INCOME) Then
   comment1 = comment1 & "Net income $" & income1
   Selection.Font.Color = wdColorRed
   Selection.TypeText (comment1)
   If (ratio1 >= MIN_RATIO) Then
       comment1 = ", " & ratio1 & "% <== BUY THIS!"
       Selection.Font.Color = wdColorBlue
       Selection.TypeText (comment1)
   End If
   Selection.TypeText (vbCr)
Else
   If (ratio1 >= MIN_RATIO) Then
         comment1 = comment1 & ratio1 & "%" & vbCr
         Selection.Font.Color = wdColorBlue
         Selection.TypeText (comment1)
     End If
End If
```

HAL: Net income $250

Spread Sheet Analyzer (5): Second Company

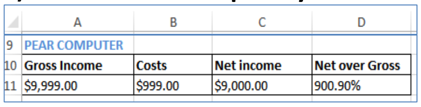


5	HAL			
6	Gross Income	Costs	Net income	Net over Gross
7	$1,500.00	$1,250.00	$250.00	20.00%

```
comment2 = company2 & ": "
If (income2 >= MIN_INCOME) Then
   comment2 = comment2 & "Net income $" & income2
   Selection.Font.Color = wdColorRed
   Selection.TypeText (comment2)
   If (ratio2 >= MIN_RATIO) Then
       comment2 = ", " & ratio2 & "% <== BUY THIS!"
       Selection.Font.Color = wdColorBlue
       Selection.TypeText (comment2)
   End If
   Selection.TypeText (vbCr)
Else
   If (ratio2 >= MIN_RATIO) Then
       comment2 = comment2 & ratio2 & "%" & vbCr
       Selection.Font.Color = wdColorBlue
       Selection.TypeText (comment2)
   End If
End If
```

PEAR COMPUTER: Net income $9000, 901% <== BUY THIS!

Spread Sheet Analyzer (6): Third Company



	A	B	C	D
9	PEAR COMPUTER			
10	Gross Income	Costs	Net income	Net over Gross
11	$9,999.00	$999.00	$9,000.00	900.90%

```
comment3 = company3 & ": "
If (income3 >= MIN_INCOME) Then
   comment3 = comment3 & "Net income $" & income3
   Selection.Font.Color = wdColorRed
   Selection.TypeText (comment3)
   If (ratio3 >= MIN_RATIO) Then
       comment3 = ", " & ratio3 & "% <== BUY THIS!"
       Selection.Font.Color = wdColorBlue
       Selection.TypeText (comment3)
   End If
   Selection.TypeText (vbCr)
Else
   If (ratio3 >= MIN_RATIO) Then
       comment3 = comment3 & ratio3 & "%" & vbCr
       Selection.Font.Color = wdColorBlue
       Selection.TypeText (comment3)
   End If
End If
```

After This Section You Should Now Know

- Objects
  - Properties/attributes vs. methods
- Using common properties/attributes and methods of the following objects
  - `Application`
  - `ActiveDocument`
  - `Selection`
- What is a named constant, why use them (benefits)
- What is a predefined named constant and what are some useful, commonly used predefined constants
- Naming conventions for constants

After This Section You Should Now Know (2)

- Collections
  - What are they
  - What is the advantage in using them
  - Common examples found in Word documents
- Using common collections in VBA
  - `Documents`
  - `Shapes`
  - `InLineShapes`
  - `Tables`
  - `Windows`

After This Section You Should Now Know (3)

- Nesting:
  - `IF` within an `IF`
  - `Do-While` within an `IF`, `IF` within a `Do-While`
  - A `Do-While` within a Do-While
  - Writing and tracing/nested structures
  - When to apply nesting

After This Section You Should Now Know (4)

- How to use the ‘`Dir`’ function to access a folder
  - Using this function to step through all the documents or specific types of documents in a folder
  - Also includes using the ‘`Len`’ function to check the length of filename and location path (String)
- Accessing other types of MS-Office programs with an VBA program written for Word

Copyright Notice