

Introduction to VBA (Visual Basic For Applications) Programming

- Origins of VBA, creating and running a VBA program
- Variables & constants
- Interactive programs
- Formatting documents
- Debugger basics
- Security
- Introduction to program documentation
- Online support: <https://support.office.com/en-US/article/create-or-run-a-macro-c6b99036-905c-49a6-818a-dfb98b7c3c9c>

B.A.S.I.C.

- Beginner's All-Purpose Symbolic Instruction Code (BASIC)
 - From: www.acm.org (original full article: <http://time.com/69316/basic/>)
- A widely used programming language
- It was relatively simple to learn (statements were "English-like" e.g., "if-then")
- Widely popular and it was commonly packaged with new computers in the 1970's and 1980's
- (A then relatively unknown company: Microsoft got it's initial cash inflows and reputation producing several versions of the language)



Visual Basic

- A newer programming language developed by Microsoft
- It was designed to make it easy to add practical and useful features to computer programs e.g., programmers could add a graphic user interface, database storage of information etc.
- Also it can take advantage of the built in capabilities of the various versions of the Windows operating system
 - Why write a feature of a program yourself when it already “comes with the computer”
- For more information:
 - <http://msdn.microsoft.com/en-us/library/2x7h1hfk.aspx>

Visual Basic For Applications (VBA)

- Shares a common core with Visual Basic.
 - Statements ‘look’ similar
- Unlike Visual Basic, VBA programs aren’t written in isolation (creating a program just for it’s own sake).
 - Most programs are written to be standalone: a computer game can be run without (say) running a web browser or MS-Office.
- VB = Visual Basic, VBA = Visual Basic for Applications
- Each VBA program must be associated with a ‘host’ *application* (usually it’s Microsoft office document such as MS-Word but other applications can also be augmented by VBA programs).
 - The host application is enhanced or supplemented by the VBA program
 - “Why doesn’t this stupid word processor have this feature??!!”
 - Now you can add that feature yourself using VBA

Visual Basic For Applications (VBA): 2

- **Important!** Because every VBA program must be run within the context of host application when you are learning to write your programs do not open up an important MS-Word document and run your programs.
 - The host program often needs an Word document in order to run certain capabilities.
 - VBA programs often change documents (formatting, style, text).
 - Therefore use only small ‘test’ MS-Word documents when running your VBA programs otherwise your information may be lost or corrupted.

Macros

- Macro: a sequence of keystrokes or mouse selections (instructions to the computer) that can be repeated over and over
 - MS-Office can be augmented by writing Macros (essentially computer programs) that will run either for multiple documents or only for a particular document.
 - In this class we will focus solely on MS-Word macro programming
- VBA (as guessed) is an example of a macro programming language e.g., you can write a program that includes a series of formatting and other commands that you frequently carry out in Word documents
- Write the commands once in the form of a program and just re-run this program instead of re-entering each command

Macros And The Web-Based Office

- According to Microsoft macros are not accessible via their online Office products:
 - <https://support.office.com/en-us/article/Differences-between-using-a-workbook-in-the-browser-and-in-Excel-f0dc28ed-b85d-4e1d-be6d-5878005db3b6?CorrelationId=917b1609-97e9-4cc7-9eeb-d188939ad740&ui=en-US&rs=en-US&ad=US>
- Result: use a computer with the desktop version of Office installed.
 - 203 lab
 - Other campus computers
 - Some 'labs' may have open access hours
 - It is CRUCIAL that you test your program on the computers in the 203 lab because your assignment must work on the lab machines in order to receive credit.

Writing Macros

- It is not assumed that you have any prior experience writing computer programs (macro language or something else).
- Consequently early examples and concepts will be quite rudimentary i.e., “we will go slow”
 - The effect is that you may find that the capabilities of the early examples will duplicate familiar capabilities already built into MS-Word
 - Why are we writing a macro program for this feature?
 - Makes it easier to understand (you know the expected result).
 - Keeps the example simpler.
- Later examples will eventually demonstrate the ‘power’ of macros
 - You can do things that would be impossible (or at least difficult) with the default capabilities built into MS-Word

Can You Complete The Following Tasks?

- Open a MS-Word document and replace every instance of one phrase e.g., tamj@ucalgary.ca with another tamj@cpsc.ucalgary.ca
- Open every document in a folder and perform the same search and replace operation:
 - 2 documents?
 - 10 documents?
 - 100 documents?
 - All the documents in a particular folder?
 - What if you just wanted to open the word documents with a particular word or phrase in the name e.g., “assignments_2014”?
- This is an example where writing a macro once is a more efficient approach
 - One answer to the question: “Why are we learning this???”

Advanced Use Of Macros

- Although it's beyond the scope of this class the following example is introduced now to make you aware of the power of VBA and macro languages.
 - It can actually be used to perform real tasks.
- You can use a macro to take advantage of the capabilities of each MS-Office application:
 - Establishing references to applications to 'link' them
 - Take the output from one application and making it the input of another.

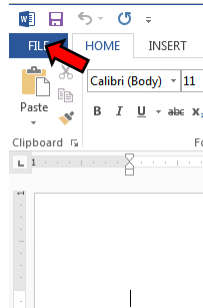
Advanced Use Of Macros (2)

- Example: macros can automate the following task
 - Store data in MS-Access
 - Store the query results in MS-Excel and perform calculations on the data
 - Use the formatting capabilities of MS-Word to produce reports
 - MS-Outlook can email the final documents

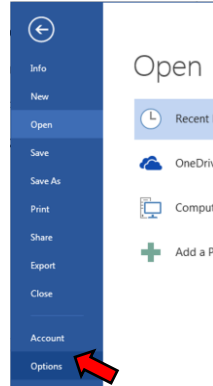
Viewing The 'Developer' Ribbon (MS-Word 2013)

- The macro programming capability comes built-in to the MS-Office suite.
 - You simply have to enable that functionality
- Steps

1. Select the 'File' ribbon



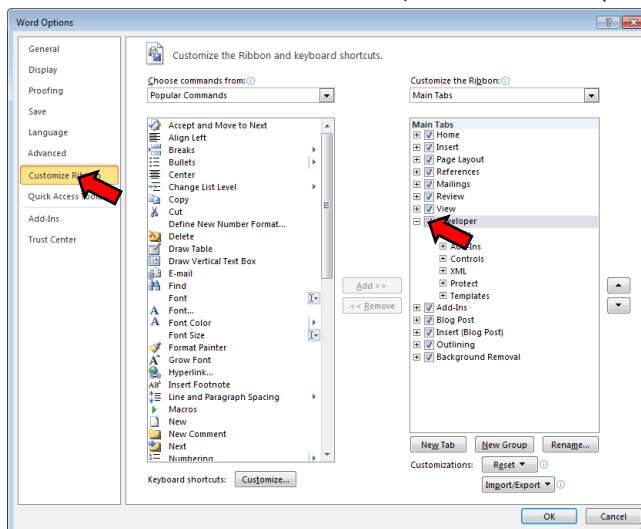
2. Select 'options'



Viewing The 'Developer' Ribbon (MS-Word 2013): 2

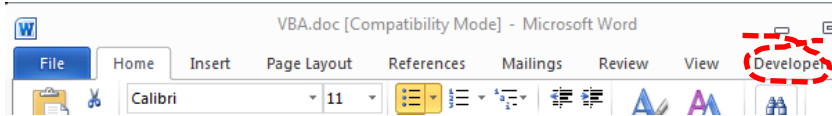
3A) Select "customize the ribbon"

3B) Check the 'Developers' box



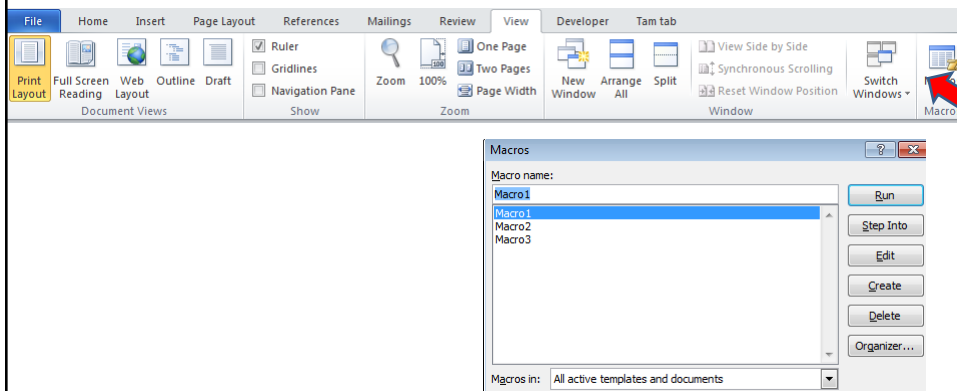
Viewing The 'Developer' Ribbon (MS-Word 2013): 3

- This should add a new ribbon "Developer"



Alternate: View And Run Macros

- You may or may not be able to edit the MS-Word ribbon with some computer labs.
- You can see view macros via the 'view' tab on the ribbon (albeit with fewer options)



Macros And Computer Security

- Computer viruses are simply malicious computer programs.
- Macros can be a useful mechanism for reducing repetition or adding new capabilities to MS-Office.
- But as is the case when writing a computer program malicious code can also be written with a macro and the virus can be activated by just opening the MS-office document that contains the macro.
- Just because you are writing macro programs does not mean that you shouldn't take macro security seriously!

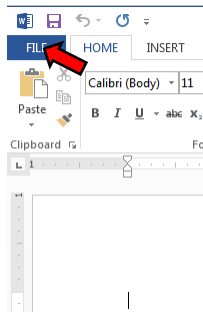
Examples Macros Viruses

- “Melissa”: Information about an old but ‘successful’ Macro Virus
 - http://www.cnn.com/TECH/computing/9903/29/melissa.02.idg/index.html?_s=PM:TECH
 - <http://www.symantec.com/press/1999/n990329.html>
 - <http://support.microsoft.com/kb/224567>
- Macro viruses aren't just “ancient history”, take the potential threat seriously!
 - <http://www.symantec.com/avcenter/macro.html>
 - <http://www.microsoft.com/security/portal/threat/encyclopedia/search.aspx?query=Virus>
 - http://ca.norton.com/search?site=nrt_n_en_CA&client=norton&q=macro+virus

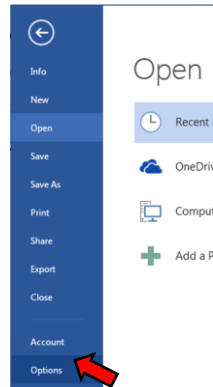
Enabling Macros To Run

- If you can't run macros in MS-office (you see odd error messages) then examine the "Trust Center" settings in Word

1. Select the 'File' ribbon



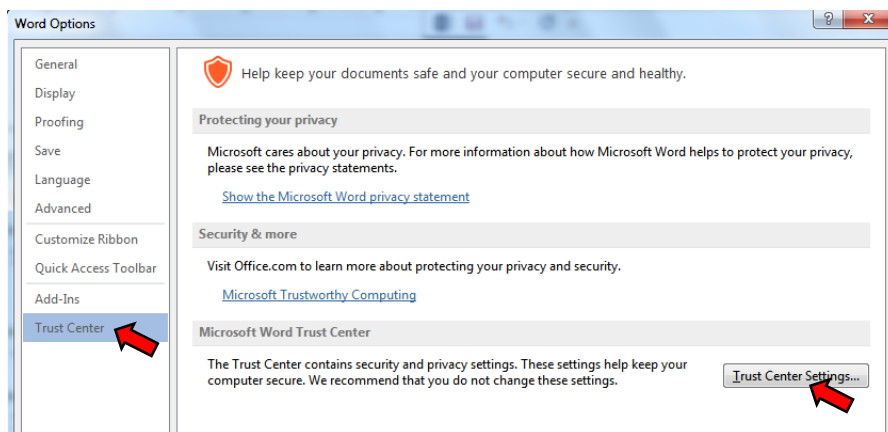
2. Select 'options'



Enabling Macros To Run (2)

3A) Select "Trust Center"

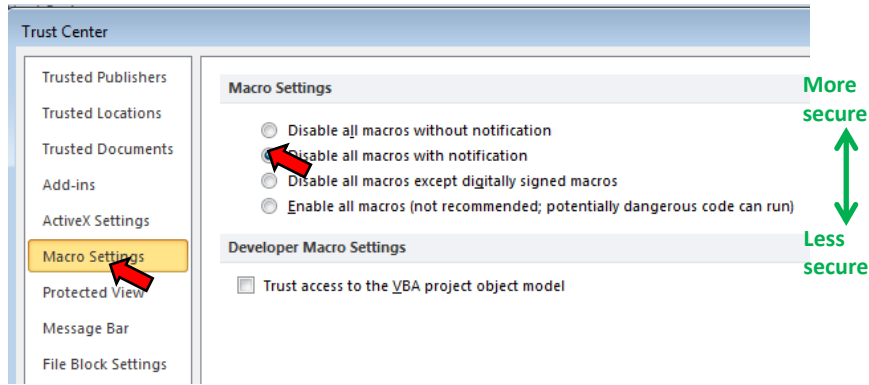
3B) Select "Trust Center Settings"



Enabling Macros To Run (3)

4A) Select "Macro Settings"

4B) Select "Disable all macros with notification"



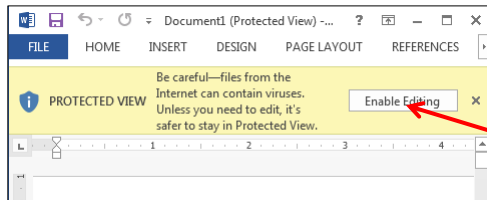
5) Exit MS-Word (close ALL documents)

Enabling Macros To Run (4)

- The default setting for MS-Word should already be set to "disable macros with notification" but these steps will allow you to use machines set differently

Effect: Opening Word Documents

- Using this setting will disable all macros by default (safer approach) but you can still enable the macros as the document is opened.

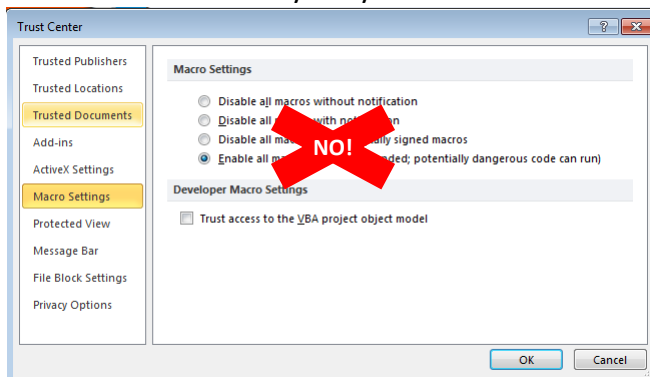


JT's caution

- You should not casually select this option for all MS-Word documents
- It's recommended that you ONLY enable macros you have created (or the lecture examples)

Macro Security

- DO NOT take the 'easy' way out



More
secure
↑
↓
Less
secure

For more information:

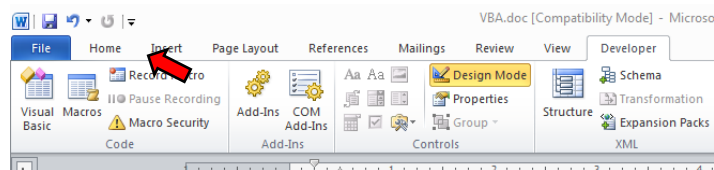
<http://www.office.microsoft.com/en-us/help/enable-or-disable-macros-in-office-documents-HA010031071.aspx>

Creating Macros

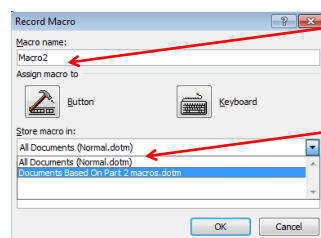
1. Record the macro automatically: keystrokes and mouse selections will be stored as part of the macro
2. Manually enter the Macro (type it in yourself into the VBA editor)

Recording Macros

- Developer ribbon
 - “Record Macro”



- Recording details



What to
name the
macro

Where to
store the
macro

Naming The Macro: Conventions

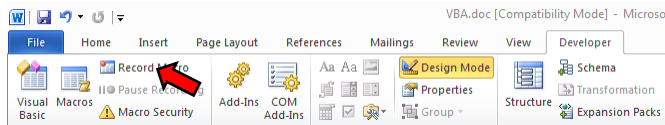
- Part of your assignment marks will be awarded according to how well your programs conform to stylistic conventions such as naming conventions employed.
 - Macros should be given a good self-explanatory name: describe purpose of the program e.g., 'formatting_resume_headings'
 - Additional information about the program can be provided in the 'description' field but for this class we will do this using 'program documentation' (described later).
- Language requirements (macro name):
 - Must start with an alphabetic letter, after than any combination of letters and numbers may be used
 - OK: "assignment1", "a2939" Not OK: "1assignment", "*assignment"
 - Maximum length of 80 characters
 - It cannot contain spaces, punctuation or special characters such as # or !
 - 'resume headings' (Not Allowed: space character)
 - 'macros!' (Not Allowed: special character)
 - Can contain underscores (separate long names)

The First Simple Macro

- With word processing there's sometimes a need to apply multiple formatting styles (bold, italics, underline) to highlighted text
- Manually applying the required formatting to each block of text can be tedious
 - Recall: Macros can be used to automate or shorten some tasks
- This first example macro program will be used to show:
 - How to create a VBA macro for MS-Word
 - How to automate a task using a macro

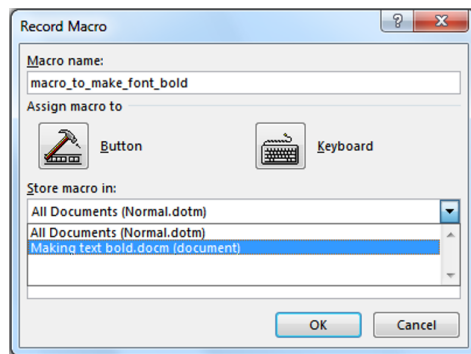
Recording A Simple Macro

- (Of course a macro isn't needed to use this formatting effect but it's easiest to start with a simple example).
- Bold face highlighted text.
 - Select the developer tab and press record



Recording A Macro (2)

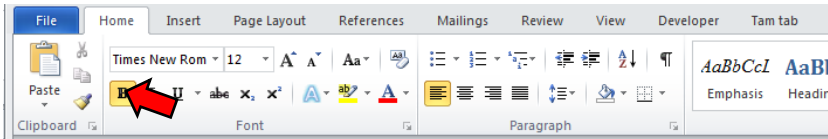
- Give the macro a self explanatory name and press 'OK' (recording begins).



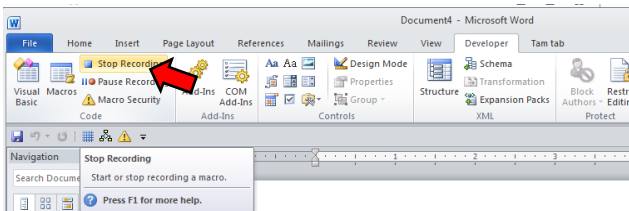
- Note: record the macro in the current document and not "All documents".

Recording A Macro (3)

- Select whatever options you want to add to the recording of the macro
 - In this case you would select bold font

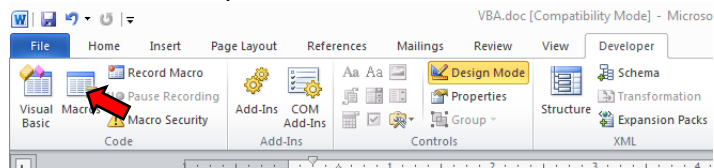


- All commands have been entered so you can stop the recording

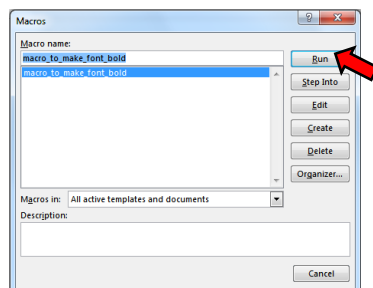


Running A Recorded Macro

- Under the Developer ribbon select 'macros'

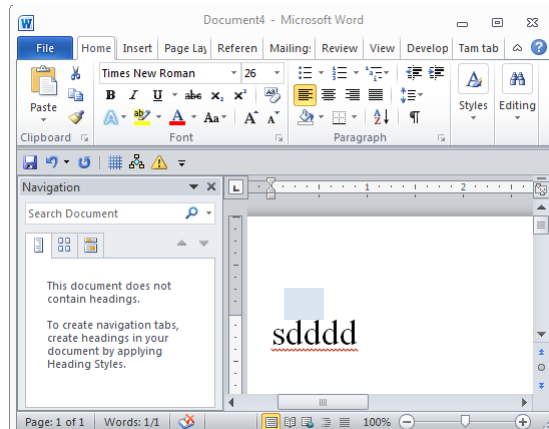


- Select the macro and then 'run' it



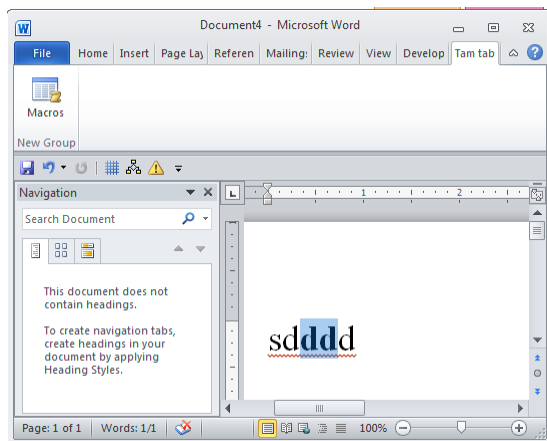
Running A Recorded Macro (2)

- In this case nothing happened?
 - This macro changes selected/highlighted text to bold
 - You need to select some text before running the macro



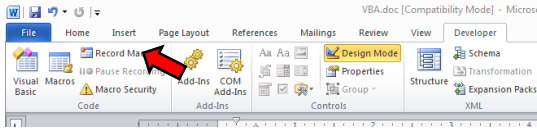
Running A Recorded Macro (3)

- After selecting the text and running the macro again, whatever text was highlighted now becomes bold.



Recording Macros: Additional Comments

- Don't rely on creating all your macros by recording them.



- Drawbacks:
 - (Problem in terms of this class) to demonstrate your understanding of concepts you will be asked to manually write VBA code
 - You won't be adequately prepared if you rely on automatically recording your programs
 - (Problem with doing this in real work) The automatically generated program code automatically is larger and more complicated than is necessary
 - "Bloated" code
 - Look under search terms such as *+bloated "vba code" +recorded* for examples of why automating recording VBA programs can be problematic.

Auto Generated VBA Program: 24 Lines

```
Sub heading()
    With Selection.ParagraphFormat
        .LeftIndent = InchesToPoints(0)
        .RightIndent = InchesToPoints(0)
        .SpaceBefore = 0
        .SpaceBeforeAuto = False
        .SpaceAfter = 6
        .SpaceAfterAuto = False
        .LineSpacingRule = wdLineSpaceSingle
        .Alignment = wdAlignParagraphCenter
        .WidowControl = True
        .OutlineLevel = wdOutlineLevelBodyText
        .CharacterUnitLeftIndent = 0
        .CharacterUnitRightIndent = 0
        .CharacterUnitFirstLineIndent = 0
        .LineUnitBefore = 0
        .LineUnitAfter = 0
        .MirrorIndents = False
        .TextboxTightWrap = wdTightNone
    End With
End Sub
```

VBA Statements *Actually Needed*: 1 Line

```
Sub headingManual()  
    Selection.ParagraphFormat.SpaceAfter = 6  
End Sub
```

Recording Macros: Additional Comments

- Benefits:
 - You can use the macro code that is automatically generated *in order to learn* how to do manually.
 - Sometimes this is very useful if you don't know the wording of a command or how to access a property.
 - Example (VBA program code for the previous example)
 - Record the commands for the macro
 - Then view the commands so you can learn how to do it manually

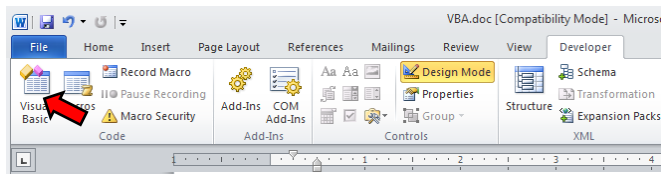
```
Sub AutoGeneratedFontChange()  
    Selection.Font.Bold = wdToggle  
End Sub
```

Recording Macros

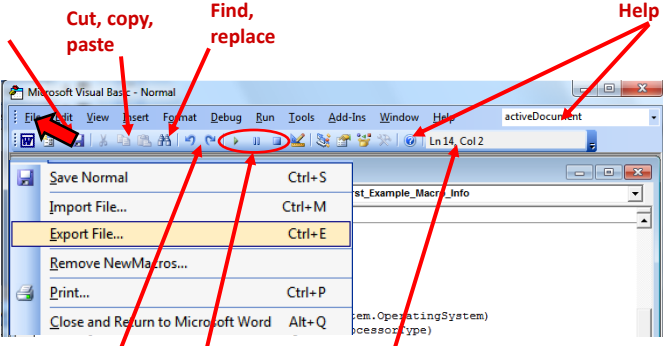
- Bottom line: use it for learning “how to do” things
- Don’t:
 - Just use the auto-generated code to study for the exam without creating any code of your code
 - Just hand in the auto-generated VBA code for your assignment
- While taking this course: Use the auto-generated code to figure out how to “type the program from scratch” yourself (I will show how to do this shortly)
- After this course is done: if ever you find your usage of Office tedious and repetitive (multiple clicks) then you can record all those steps into one macro!

The Visual Basic Editor

- You don’t need to familiarize yourself with every detail of the editor in order to create VBA programs.
- Just a few key features should be sufficient
- Starting the editor:
 - Because VBA programs are associated with an office application open the editor from MS-Word
 - Click the “Visual Basic” icon under “Developer”



Overview Of The Important Parts Of The VBA Editor



Save

Cut, copy, paste

Find, replace

Help lookup

Undo, redo

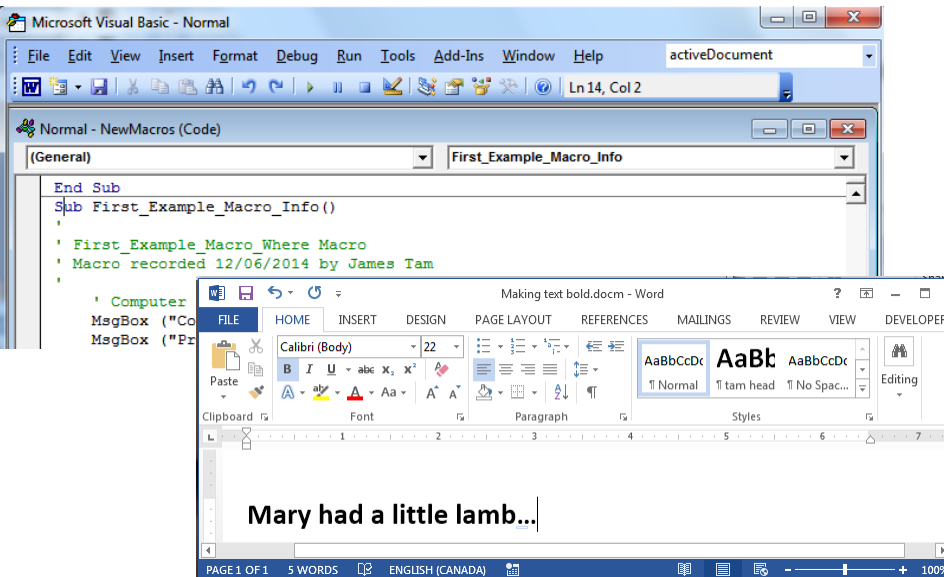
Run, pause, stop (VBA subroutine program)

Current location

Program editor

Export:
Useful for transferring or backing up your work

VBA Editor: Don't Mix It Up With The Word Editor



Microsoft Visual Basic - Normal

Microsoft Word - Making text bold.docm

```

End Sub
Sub First_Example_Macro_Info()
    ' First_Example_Macro Where Macro
    ' Macro recorded 12/06/2014 by James Tam
    ' Computer
    MsgBox ("Pr...
    MsgBox ("Pr...
  
```

Mary had a little lamb...

Defining A Program In The VBA Editor

- **Format:**

```
' Program documentation goes here (more on this later)
sub <sub-program name>()
    Instructions in the body of program (indent 4 spaces)
End Sub
```

- **Example:**

```
' Author, version, features etc.
Sub first_example_macro_info()
    MsgBox ("Congratulations! This your first computer
           program")
End Sub
```

- Note: large VBA programs have multiple (sub) parts but for this class you only need to define a single 'sub'.

The 'Sub' Keyword

- Sub stands for 'subroutine' or a portion of a VBA program

Format:

```
Sub <subroutine name>()
    <Instructions in the subroutine>
End Sub
```

Header, start of subroutine:

- Has word 'Sub'
- Name of subroutine
- Set of brackets

Note: all lines in between are indented (4 spaces)

- **Example:**

```
Sub First_Example_Macro_Info()

End Sub
```

End of subroutine:

- Has 'End Sub'

- All executable VBA program statements must be inside the subroutine

The 'Sub' Keyword: 2

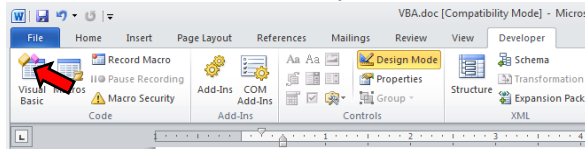
- Real world VBA programs will be broken down into multiple 'subs' (subroutines or program parts)
- Because this is only brief introduction into writing VBA programs you will only have to define one subroutine for your assignment.

VBA Examples: This Point Onwards

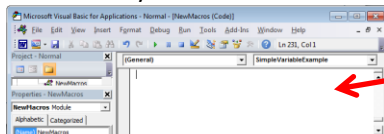
- Unlike the previous example you will be manually typing in the program instructions yourself rather than automatically recording the program as a series of steps
- Reminder: some of the early examples are meant only as a learning/teaching tool
 - They show you how to write simple VBA programs
 - So they won't yet focus on "doing useful things" yet
- Try typing them into the VBA editor or cutting and pasting them yourself
 - It's important to "try things out for yourself"
 - With programming you learn by "doing yourself" rather than by watching someone else 'do'

First VBA Example

- Learning Objectives:
 - Creating/running a VBA program
 - Creating a Message Box “MsgBox”
- Reminder steps (since this is your first example)
 - Start up the application (MS-Word)
 - Invoke the VBA editor: Developer->Visual Basic



- If successful you should see something similar to the image

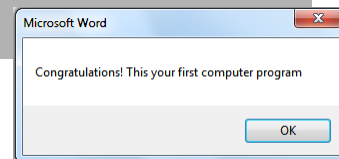


Enter your program instructions here (program editor)

First VBA Example (2)

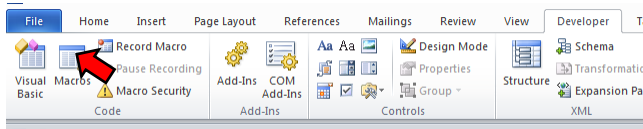
- Type in or cut-and-paste the following example into the *VBA editor* (see last image for location of the editor, previous slide)
 - This is **NOT** the same as pasting it directly into MS-Word.
 - **Word document containing the macro: 1firstExampleMacro.docm**

```
Sub first_example_macro_info()
    MsgBox ("Congratulations! This your first computer
           program")
End Sub
```

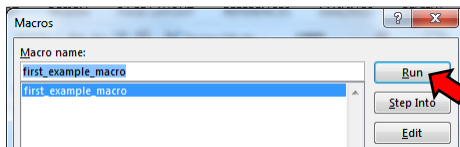


Reminder: Running Macros

- (You must first have the 'developer' tab visible).
- Developer->Macros

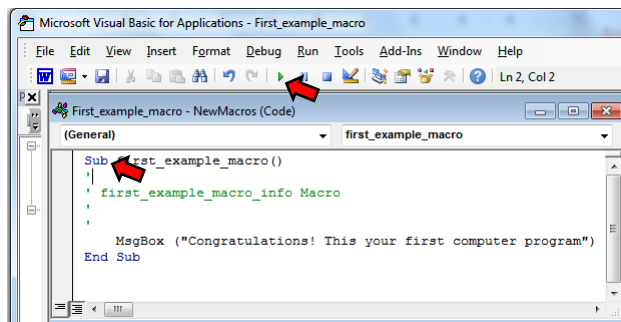


- The single macro should be highlighted, then click 'run'



Running VBA Programs You Have Entered (2)

- Or you can run the program right after you have entered it (in the editor).



1. Ensure correct program "sub" is to be executed (click there)
2. Press the 'play/run' button or "F5"

Structure Of VBA Programs: Reminder Of Important Points

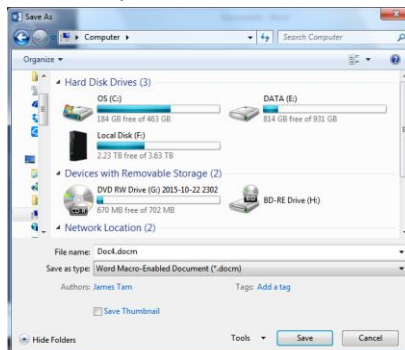
- As you just saw a program must begin with the “sub” keyword followed by the name of the “subroutine” (sub-part of the program).
- It also ends with end “end sub”
- Important style requirement: The part between the ‘sub’ and ‘end sub’ must be indented by 4 spaces (8 spaces if sub-indenting is used – next set of notes).

```
sub first_example_macro()
    MsgBox("Congrats!")
end Sub
```

Saving Your Macro

You can save your macro into a Word document:

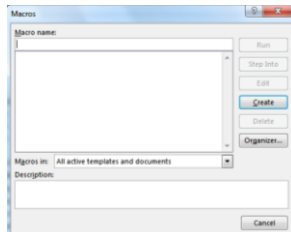
1. Create a new Word document
2. Save the document as a macro-enabled document (Word document that has a macro computer program embedded within it).



Saving Your Macro (2)

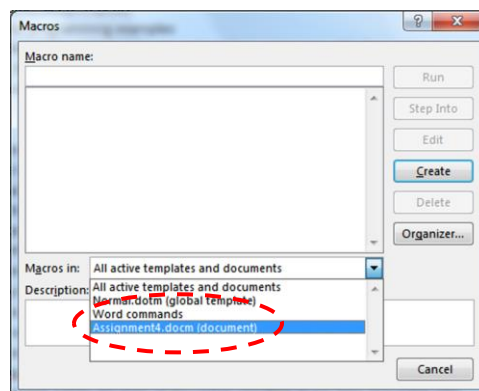
3. Next you have tell Word that you want to save your macro program inside this macro-enabled document.

- By default when you save your macros Word will select “Normal.dotm” as the location.
- DO NOT save your macro in this document:
 - You will have trouble transferring your macro to other computers
 - Because “Normal.dotm” is the default template used to create some Word documents it may result in security warnings.



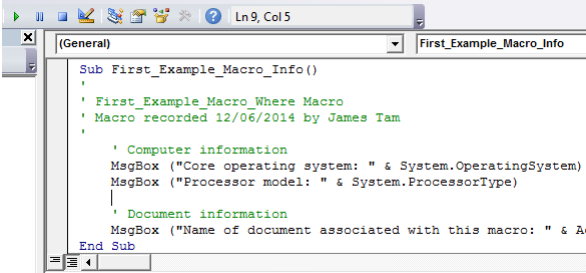
Saving Your Macro (3)

- Instead: Save your macro in your current document (in the example below it's “Assignment4.docm”).
- Transferring this document will allow the macros to be transferred as well.



Viewing Macros

- All macros that you have created can be viewed in the VBA macro editor:
 - Macros manually entered in the editor (Message Box example)
 - Macros automatically recorded (previous example: changing font to bold)








```

Sub First_Example_Macro_Info()
' First_Example_Macro_Where Macro
' Macro recorded 12/06/2014 by James Tam
'
' Computer information
MsgBox ("Core operating system: " & System.OperatingSystem)
MsgBox ("Processor model: " & System.ProcessorType)
'
' Document information
MsgBox ("Name of document associated with this macro: " & A
End Sub

```

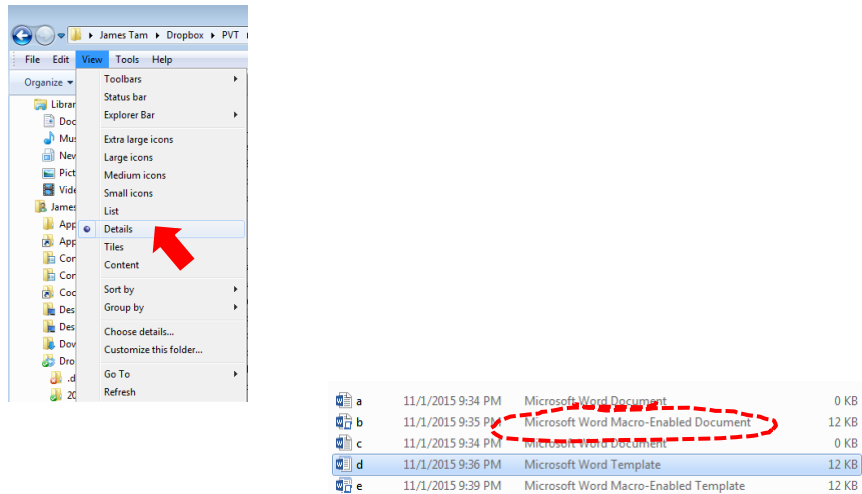
Previous example: auto recorded

Which Document Contains A Macro?

 a	11/1/2015 9:34 PM	Microsoft ...	0 KB
 b	11/1/2015 9:35 PM	Microsoft ...	12 KB
 c	11/1/2015 9:34 PM	Microsoft ...	0 KB
 d	11/1/2015 9:36 PM	Microsoft ...	12 KB
 e	11/1/2015 9:39 PM	Microsoft ...	12 KB

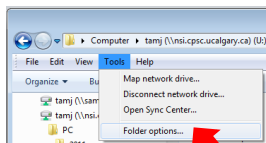
Viewing File Information

- View details: select 'view' in a folder

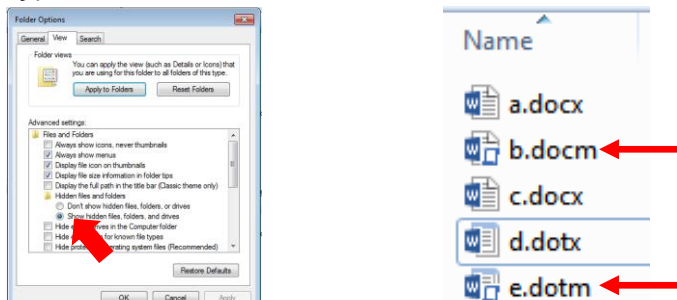


Viewing File Suffixes

- In a folder select: Tools->Folder options

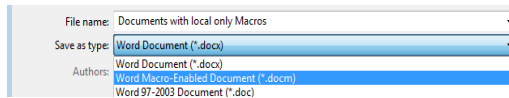


- Under the 'view' tab uncheck 'Hide extensions for known file types'



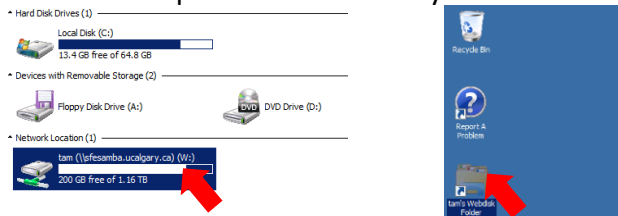
Transferring Your Macros (This Class)

- If you create a macro-enabled MS-Word document (file name suffix “.docm”) then transfer the Word document itself.



Transferring Your Macros (This Class): 2

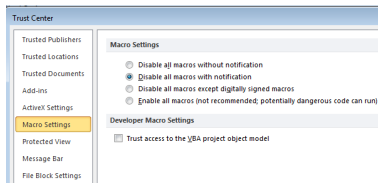
- In the 203 computer lab
- Save the macro enabled word document to your portable USB flash drive
- OR
- Save the template document on your web disk drive



- To download files stored on web disk onto another computer:
<https://webdisk.ualgary.ca/>

More On Macro Security

- A macro is a computer program that is attached to another program's documents (e.g., MS-Word documents)
- It can supplement the program's features by automating repetitive tasks
- But like another computer program the instructions can either be useful or malicious
- Security settings are specified in the MS-Office "Trust Centre"



Types Of Documents That Can Contain Macros


- You store the macros that you write for this class this way
 - In a single document 'doc-m' document
- You can also store macros in these documents (not for this class but important to be aware in terms of computer security).
 - Normal 'dot-m' template
 - Custom 'dot-m' template


Question: What Is The Security Difference?

- Opening the following documents:
 - Document.docm
 - Document.docx
 - Document.doc

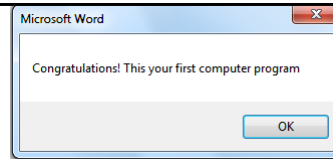
.DOCX (And .XLSX, .PPTX)

- As mentioned these types of files cannot have macros attached to them.
 - Reduced capabilities (no macros) but increased security (no macros)
- Question: Are these files with these extensions 100% safe?

 Trust me - I'm safe.docx

 Trust me - I'm safe.docx.doc

Message Box



- (Details of the previous example)
- Creates a popup window
- Useful for testing
 - Is my program working?
 - Which part is running?
- Also useful for displaying status messages about the current state of the program

Creating A **Message Box**

- **Format:**
MsgBox (*"<Message to appear>"*)
- **Example:**
MsgBox ("Congratulations! This your first computer program")

Notes on 'Format':

- Italicized: you have a choice for this part
- Non-italicized: mandatory (enter it as-is)
- Don't type in the angled brackets (used to help you visually group)

VBA Visual Aids: **Function Arguments**

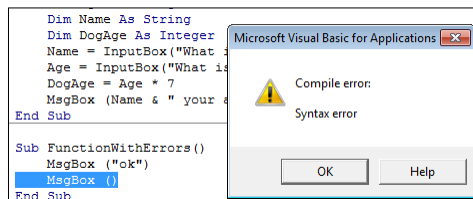
- As you type in the name of VB functions you will see visual hints about the arguments/inputs for the function.
 - Enter the function name and then a space

```
Sub FunctionWithErrors ()
  MsgBox (
En MsgBox(Prompt, [Buttons As VbMsgBoxStyle = vbOKOnly], [Title], [HelpFile], [Context]) As VbMsgBoxResult
```

Function arguments

(**Bold**): mandatory arguments

VBA Visual Aids: **Error Information**



Required argument missing

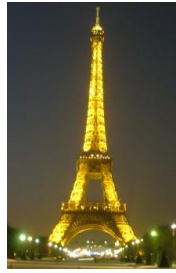
Part of program that contains errors (yellow highlight)

```
Sub FunctionWithErrors ()
  MsgBox ("ok")
  MsgBox ()
End Sub
```

Specific statement/instruction causing the error (red font)

Real-World Objects

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



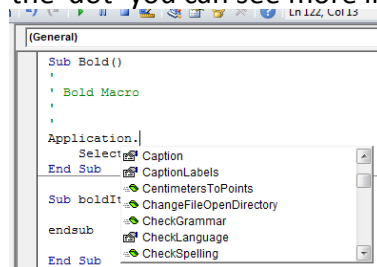
- Each object is described by its properties (information)
- Each object can have a set of functions associated with it (actions)

VBA Object

- 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 etc.
 - Capabilities: actions that can be performed (sometimes referred to as 'methods' or 'functions')
 - E.g., save, print, spell check etc.

Common VBA Objects

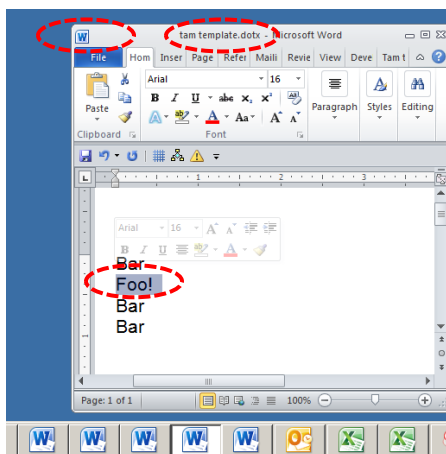
- **Application**: the MS-Office program running (for CPSC 203 it will always be MS-Word)
- **ActiveDocument**
- **Selection**
- When enter one of these keywords in the editor followed by the 'dot' you can see more information.



Take advantage of the benefits:

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*
- Current Document:
 - *"tamj template"*
- Selection
 - *"Foo!"*

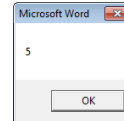
Using Pre-Built Capabilities/Properties Of Objects

- **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'

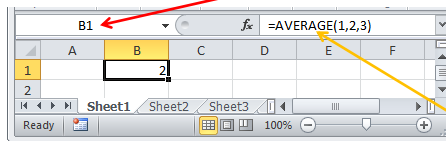
Accessing the Windows property of Word (the application)
• Info about the windows currently opened

Properties Vs. Methods/Functions

- Recall

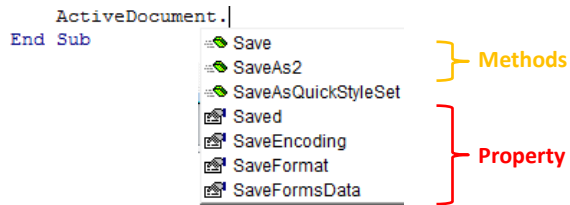
- **Property:** information about an object
- **Method:** capabilities of an object (possible actions)

Property:
current cell



Using the
'average()' function

Properties Vs. Methods: Appearance



- Similar to functions in MS-Excel some object's methods may require an argument or arguments

- Examples

- ActiveDocument.CountNumberedItems ← No argument required
- ActiveDocument.Save ← No argument required
- ActiveDocument.SaveAs2("<name>") ← Argument: New name of document needed

More On Objects

- This is only a very brief introduction on VBA objects
- What you should now know
 - The 3 common objects (application, active document, selection)
 - Objects consist of attributes and methods and how to access/use them
- As the need arises you will learn more about these objects (and perhaps others) in the next section

Basic Mathematical **Operators**

Operation	Symbol used in VBA	Example
Addition	+	2 + 2
Subtraction	-	3 - 2
Multiplication	*	10 * 10
Division	/	81 / 9
Exponent	^	2 ^ 3

Variables

- Used to temporarily store information at location in memory
- Variables must be declared (created) before they can be used.
- **Format for declaration:**
Dim <Variable name> as <Type of variable>
- **Example declaration:**
Dim BirthYear as Long

Image courtesy of James Tam

Some Types Of Variables

Type of information stored	VBA Name	Example variable declaration	Default Value
Whole numbers	Long	Dim LuckyNumber as Long	0
Real numbers	Double	Dim MyWeight As Double	0
Characters ¹	String ²	Dim Name As String	Empty string
Date ³	Date	Dim BirthDate As Date	00:00:00

- 1) Any visible character you can type and more e.g., 'Enter' key
- 2) Each string can contain up to a maximum of 2 billion characters
- 3) Format: Day/month/year

Examples Of Assigning Values To Variables

Note: some types of variables requires some mechanism to specify the type of information to be stored:

- Strings: the start and end of the string must be marked with double quotes "
- Date: the start and end of the string must be marked with the number sign #

```
Dim LuckyNumber As Long
LuckNumber = 888
```

```
Dim BirthDay As Date
BirthDay = #11/01/1977#
```

```
Dim MyName As String
MyName = "James"
```


Variables: Metaphor To Use



- Think of VBA variables like a “mailslot in memory”
- Unlike an actual mailslot computer variables can only hold one piece of information
 - Adding new information results in the old information being replaced by the new information

```
Dim num as Long
```

num



```
num = 1
```

num



```
num = 17
```

Variables: Metaphor To Use (2)

- Also each computer variable is separate location in memory.
- Each location can hold information independently of other locations.
- Note: This works differently than mathematical variables!

```
Dim num1 as Long
Dim num2 as Long
num1 = 1
num2 = num1
num1 = 2
```

- What is the result?

MsgBox: Displaying Mixes Of Strings And Variables

- **Format:**

```
MsgBox ("<Message1>" & <variable name>)
```

- **Example:**

```
Dim num as Long
```

```
num = 7
```

```
MsgBox ("num=" & num)
```

```
"num="      : A literal string
```

```
num:        : contents of a variable (slot in memory)
```

Second Example: Basics Of Variables

- Name of the online example: `2variablesMixedOutput.docm`

```
Sub secondExample()
  Dim num1 As Long
  Dim num2 As Long
  num1 = 1
  num2 = num1
  MsgBox ("num1=" & num1 & ", " & "num2=" & num2)
  num1 = 2
  MsgBox ("num1=" & num1 & ", " & "num2=" & num2)
End Sub
```

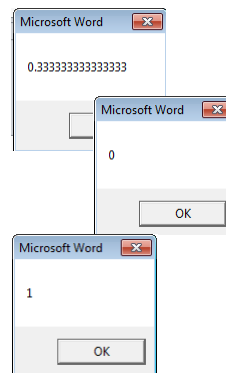
Third VBA Example

- Learning Objectives:
 - Using variables
 - Using mathematical operators

Third VBA Example (2)

Word document containing the macro: 3types.docm

```
Sub thirdExample()  
    Dim RealNumber As Double  
    Dim WholeNumber As Integer  
  
    RealNumber = 1 / 3  
    MsgBox (RealNumber)  
    WholeNumber = 5 / 10  
    MsgBox (WholeNumber)  
    WholeNumber = 6 / 10  
    MsgBox (WholeNumber)  
End Sub
```



JT's note: Anything over 0.5 is rounded up

Student Exercise #1

- What would appear as output (MsgBox) when the following VBA macro program was executed

```
Sub exercise1()  
  Dim num1 As Long  
  num1 = 1  
  MsgBox (num1)  
  num1 = num1 + 1  
  MsgBox (num1)  
End Sub
```

Student Exercise #2

- What would appear as output (MsgBox) when the following VBA macro program was executed

```
Sub exercise2()  
  Dim num1 As Long  
  Dim num2 As Double  
  
  num1 = 10  
  num2 = 10.5  
  
  MsgBox (num1 & "---" & num2)  
  
  num1 = num2 + 10  
  num2 = num1 * 2 / 3  
  MsgBox (num1 & "---" & num2)  
  
End Sub
```

Variable Naming Conventions

- Language requirements:
 - Rules built into the Visual Basic (recall VBA is essentially Visual Basic tied to an MS-Office Application) language.
 - Somewhat analogous to the grammar of a 'human' language.
 - If the rules are violated then the typical outcome is the program cannot execute.
- Style requirements:
 - Approaches for producing a well written program.
 - (The real life analogy is that something written in a human language may follow the grammar but still be poorly written).
 - If style requirements are not followed then the program can execute but there may be other problems (e.g., it is difficult to understand because it's overly long and complex - more on this during the term).

Naming Variables: VBA Language Requirements

- Names **must** begin with an alphabetic character
 - OK: name1 Not OK: 1name
- Names **cannot** contain a space
 - OK: firstName Not OK: first name
- Names **cannot** use special characters anywhere in the name
 - Punctuation: ! ? .
 - Mathematical operators: + - * / ^
 - Comparison operators: < <= > >= <> =

Naming Variables: Style Conventions

- | | Examples |
|--|---|
| 1. Style requirement (all languages): The name should be meaningful. | #1:
age (yes)
x, y (no) |
| 2. Style requirement (from the Microsoft Developer Network ¹): | |
| a) Choose easily readable identifier names | HorizontalAlignment (yes)
AlignmentHorizontal (no) |
| b) Favor readability over brevity. | CanScrollHorizontally (yes)
ScrollableX (no) |

¹ <http://msdn.microsoft.com/en-us/library/ms229045.aspx>

Naming Variables: Style Conventions (2)

- | | Examples |
|---|--|
| 3. Style requirement: Variable names should generally be all lower case except perhaps for the first letter (see next point for the exception). | #3:
age, height, weight (yes)
HEIGHT (no) |
| 4. Style requirement: For names composed of multiple words separate each word by capitalizing the first letter of each word (save for the first word) or by using an underscore. (Either approach is acceptable but don't mix and match.) | #4
firstName, last_name
(yes to either approach) |
| 5. Avoid using keywords as names (next slide) | |

Some Common Visual Basic Keywords¹

And	Boolean	Call	Case	Catch	Continue
Date	Decimal	Default	Dim	Do	Double
Each	Else	End	Erase	Error	Event
Exit	False	Finally	For	Friend	Function
Get	Global	Handles	If	In	Inherits
Integer	Interface	Is	Let	Lib	Like
Long	Loop	Me	Mod	Module	Next
Not	Nothing	Of	On	Operator	Option
Optional	Or	Out	Overrides	Partial	Private
Property	Protected	Public	Resume	Return	Select
Set	Shadows	Short	Single	Static	Step
Stop	String	Sub	Then	Throw	To
True	Try	Using	Variant	When	While
Widening	With				

¹ The full list can be found on the MSDN <http://msdn.microsoft.com/en-us/library/dd409611.aspx>

Variable Naming Conventions: Bottom Line

- Both the language and style requirements should be followed when declaring your variables.


Obtrusive Popup Windows/Messages

- Great! I've got the info that I need.

How Does a Turbo Charger Work?

By John Albers, eHow Contributor

Like Share 4 Tweet 0 Share Pin It



Other People Are Reading

- What Are the Functions of a Turbo Charger?
- How a Variable Vane Turbo Charger Works

Purpose

A turbo charger is used in high performance vehicles and can be added as an after-market option to most cars as a cheap and energy efficient method of increasing an engine's power output. It

TOMORROW starts here. CISCO WebEx Meetings Premium

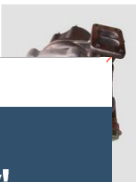
Obtrusive Popup Windows/Messages

- Hey I was reading that

How Does a Turbo Charger Work?

By John Albers, eHow Contributor

Like Share 4 Tweet 0 Share Pin It



eHowNow

Have a tech question?
Ask online tech support now!

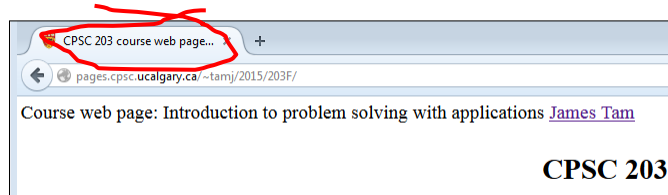
Type Your Question Here...

Get an Answer

energy efficient method of increasing an engine's power output. It

TOMORROW starts here. CISCO WebEx Meetings Premium

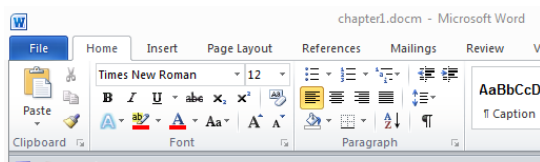
Alternate To Using Popup Messages



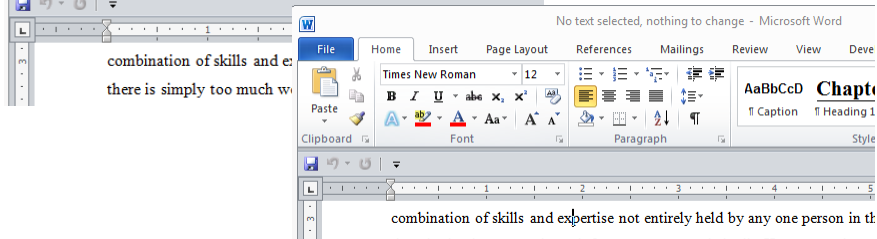
An Alternate To The MsgBox: The Title Bar

- Finding the MsgBox popups too obtrusive?
- You can display messages in a more subtle fashion by changing the title bar.

Before



After



Changing The Title Bar

- **Format:**

```
ActiveDocument.ActiveWindow.Caption =
    "<New text for the title bar>"
```

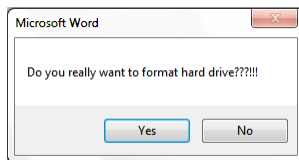
- **Word document containing the macro: 4titleBar.docm**

```
Sub firstTitleBarExample()
    ActiveDocument.ActiveWindow.Caption = "A NEW TITLE!"
End Sub
```

Comparison: Popups Vs. Status Messages

- **Popup dialogs:**

- Often used for important messages that you don't want the user to miss



- Overuse can simply result in the user "clicking past" the dialog without reading them

- **Status messages:**

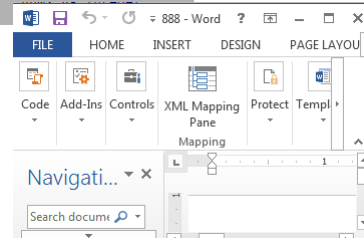
- More subtle presentation of information.
- Easy to miss.
- Multiple updates result in only the last change appearing to the user

Missing The Message

Word document containing the macro: 5titleBarV2.docm

```
Sub secondTitleBarExample()
    Dim num1 As Integer
    Dim num2 As Integer

    num1 = 666
    num2 = 888
    ActiveDocument.ActiveWindow.Caption = num1
    ActiveDocument.ActiveWindow.Caption = num2
End Sub
```

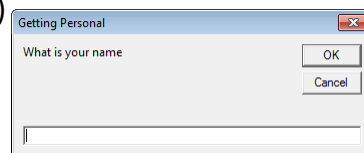


Getting **User Input**

- A simple approach is to use an **Input Box**
- Format:


```
<Variable name> = InputBox(<"Prompt">, <"Title bar">)
```
- Example:

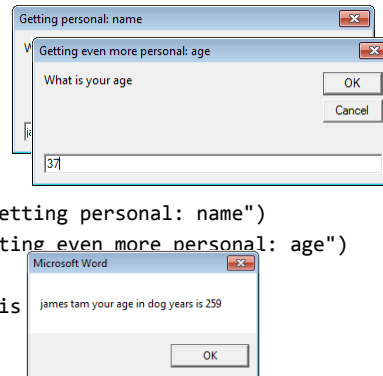

```
Name = InputBox("What is your name"), "Getting Personal")
```
- Note: only the string for the prompt is mandatory.
- If the title bar information is omitted then the default is the application name ("Microsoft Word")



Example: InputBox

- Learning: getting user input with an InputBox
- **Word document containing the macro:**
6inputBox.docm

```
Sub InputExample()
    Dim Age As Integer
    Dim Name As String
    Dim DogAge As Integer
    Name = InputBox("What is your name", "Getting personal: name")
    Age = InputBox("What is your age", "Getting even more personal: age")
    DogAge = Age * 7
    MsgBox (Name & " your age in dog years is " & DogAge)
End Sub
```



Note: there are two input boxes, one that prompts for the name and the other for the age. Each is given a *self-descriptive name* to distinguish them (an example of *good programming style* – more on this shortly)

The VBA Debugger

- Debuggers can be used to help find errors in your program
- Setting up breakpoints
 - Points in the program that will ‘pause’ until you proceed to the next step
 - Useful in different situations
 - The program ‘crashes’ but you don’t know where it is occurring
 - Pause before the crash
 - An incorrect result is produced but where is the calculation wrong
- Set up breakpoints
 - Click in the left margin

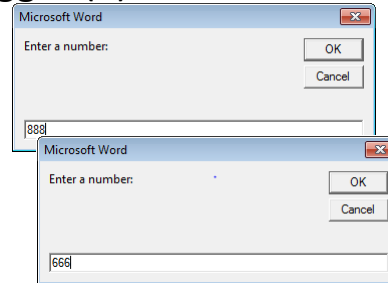
```
Sub debugExample()
    Dim numerator As Long
    Dim denominator As Long
    Dim quotient As Double
    numerator = InputBox("Enter a number")
    denominator = InputBox("Enter a number")
    quotient = numerator / denominator
    MsgBox (quotient)
End Sub
```

The VBA Debugger (2)

- Multiple breakpoints

```
Sub DebugExample ()
    Dim Double1 As Double
    Dim Double2 As Double
    Dim Double3 As Double

    Double1 = InputBox("Enter a number: ")
    Double2 = InputBox("Enter a number: ")
    Double3 = Double1 / Double2
End Sub
```



- Program pauses when breakpoints are reached
 - The contents of variables can be displayed at that point in the program

```
Sub DebugExample ()
    Dim Double1 As Double
    Dim Double2 As Double
    Dim Double3 As Double

    Double1 = InputBox("Enter a number: ")
    Double2 = InputBox("Enter a number: ")
    Double3 = Double1 / Double2
End Sub
```

Expression	Value	Type
NewMacros	0	NewMacros/NewMacros
Double1	0	Double
Double2	0	Double
Double3	0	Double

```
Sub DebugExample ()
    Dim Double1 As Double
    Dim Double2 As Double
    Dim Double3 As Double

    Double1 = InputBox("Enter a number: ")
    Double2 = InputBox("Enter a number: ")
    Double3 = Double1 / Double2
End Sub
```

Expression	Value	Type
NewMacros	888	NewMacros/NewMacros
Double1	888	Double
Double2	666	Double
Double3	0	Double

The VBA Debugger (3)

- Because the VBA debugger is interactive it's best to see the results live in "real time"
- So you will see more on the debugger in tutorial

Break Points: Final Note

- They only meant as a temporary mechanism for finding the errors in your program.
 - (Having them 'permanently' included with the program would be a nuisance because it will pause at each break point).
- Consequently break points are not saved either with: the document, template or the VBA program

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 so 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

- **Example:**

```
Sub ConstantExample()
    Const PI = 3.14
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

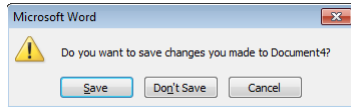
Yes 😊

Predefined Constants: MS-Word

- Microsoft uses their own naming convention
- Example:
 - wdPromptToSaveChanges
- Usage:
 - `ActiveDocument.Close(wdPromptToSaveChanges)`

Closing Documents

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



- VBA code to close a document in this fashion:

```
ActiveDocument.Close (wdPromptToSaveChanges)
```

Pre-defined constant

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

- **Word document containing the macro:**
"7closingActions.docm"

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

'Choose one constant
wdPromptToSaveChanges
wdDoNotSaveChanges
wdSaveChanges

```
End Sub
```

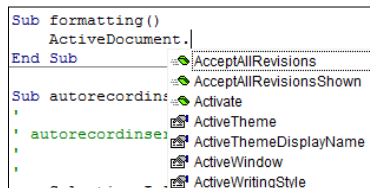
Formatting Text

- Allows formatting changes to be applied to the text in a Word document:
 - Examples: bold, underline, font type, font size
- Some approaches:
 - Apply the formatting changes to the entire document
 - Apply the formatting changes to select parts of the document

Formatting An Entire Document

- You first need to indicate the document to be formatted
- This can be done through the 'ActiveDocument' object

```
Sub formatting()
  ActiveDocument.|
End Sub
Sub autorecordins:
  ' autorecordins:
  ' 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

Active Document: Which One?

- Remember: the active document is the one you are currently working on.
- When writing VBA programs this can be tricky to determine.



- So it's best to work with only a single document at a time when writing your programs.

Formatting Text: 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 = Calibri
```

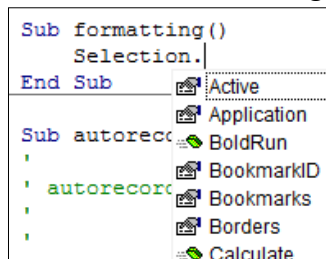
- **Word document containing the macro:**
8formattingEntireDocument.docm

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

Formatting The Text Currently Selected

- This can be done through the 'Selection' object

```
Sub formatting()  
    Selection.  
End Sub
```



- This object accesses the text in a document that has been selected

Changing Selected Text

- **Word document containing the macro: 9cutAndPaste.docm**
 - Learning concepts: editing selected text, cutting and pasting text, using constants

```
Sub cutAndPaste()
  Selection.Expand
  Selection.Copy
  Selection.MoveRight
  Selection.PasteAndFormat
  (wdFormatOriginalFormatting)
End Sub
```

The diagram shows the text "The cat in the hat." in three stages of the macro's execution:

- Initial state: "The cat in the hat." (The word "cat" is underlined).
- After Selection.Expand and Selection.Copy: "The ~~cat~~ in the hat." (The word "cat" is crossed out).
- After Selection.MoveRight and Selection.PasteAndFormat (wdFormatOriginalFormatting): "The cat cat in the hat." (The word "cat" is underlined and repeated).

Below the code, a red bracket highlights the constant `wdFormatOriginalFormatting` in the code, with the following notes:

- A predefined constant
- Another is "wdFormatPlainText"

On the right, a dropdown menu for the `PasteAndFormat` method is shown, listing various constants:

- wdChart
- wdChartLinked
- wdChartPicture
- wdFormatOriginalFormatting
- wdFormatPlainText
- wdFormatSurroundingFormattingW
- wdListCombineWithExistingList

Student Exercise #3

- What would happen to the following document...

The *cat in the hat.*

- ...if the following macro was executed?

```
Sub exercise2()
  Selection.Expand
  Selection.Copy
  Selection.MoveRight
  Selection.PasteAndFormat (wdFormatPlainText)
End Sub
```

Formatting Selected Text: Multiple Changes

- Desired formatting:
 - Font size = 14 point
 - Bold
 - Underline
 - Center the heading
 - 6 point spacing after the heading
- **Word document name:**
10formattingSelectedTextMultiple.docm

```
Sub formattingHeadings()
    Selection.Font.Size = 14
    Selection.Font.Bold = True
    Selection.Font.Underline = True
    Selection.ParagraphFormat.Alignment =
        wdAlignParagraphCenter
    Selection.ParagraphFormat.SpaceAfter = 6
End Sub
```

Effect

- Before:

```
Absorb what is useful,
reject what is useless,
add what is specifically your own.
--
Bruce Lee
```

- After

```
Absorb what is useful,
reject what is useless,
add what is specifically your own.
--
Bruce Lee
```

Formatting Selected Paragraphs

- Desired formatting:
 - Font size = 10 point
 - Left and right margins indented by 0.2 inches
- **Word document:** 11formattingSelectedParagraphs.docm

```
Sub formattingParagraphs()
    Selection.Font.Size = 10
    Selection.Font.Bold = True
    Selection.ParagraphFormat.LeftIndent = InchesToPoints(0.2)
    Selection.ParagraphFormat.RightIndent = InchesToPoints(0.2)
End Sub
```

Le Marais: An interesting place, apparently it was once a swamp and then it became the center of high culture in Paris which then fell into disrepair after the French Revolution. Now it looks like a place to stay, there are many shops, cafes and the like here. It's closer to Notre Dame on the other side of the river but if I were to stay in Paris a second time I would think of staying here.

Here's a picture of one of the many cafes where you can just sit and watch life in



Le Marais: An interesting place, apparently it was once a swamp and then it became the center of high culture in Paris which then fell into disrepair after the French Revolution. Now it looks like a place to stay, there are many shops, cafes and the like here. It's closer to Notre Dame on the other side of the river but if I were to stay in Paris a second time I would think of staying here.

Here's a picture of one of the many cafes where you can just sit and watch life in the streets go by.



What's The Difference?

```
' First example
Selection.Font.Bold = wdToggle
```

```
' Part of the example just covered
Selection.Font.Bold = True
```

Advanced Concept: What If there Is No Selection?

- The VBA program attempts to format the currently selected text but there is 'no' selection.
- (Rhetorical questions - for now)
 - What will happen?
 - What modifications can be made to the program to handle this case?

Formatting: Recap

- Formatting the entire document: access the **ActiveDocument** object
- ```
ActiveDocument.Select
```
- Formatting the currently selected text: access the **Selection** object
  - Then select the property of the selection that you wish to set/change
  - Example: changing font of selected text
    - 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 = Calibri
```



## Printing: Single

- Printing a single document (currently opened, active MS-Word document)
- **Word document containing the macro example:**  
"12singleDocumentPrint.docm"

```
Sub PrintSingleDocument()
 ActiveDocument.PrintOut
End Sub
```

## Program Documentation

- Your VBA assignment submission must include identification information:
  - Full name
  - Student identification number
  - Tutorial number
  - List the program features (from the assignment description) and clearly indicate if the feature was completed or not completed.
- DON'T just enter this information into your program instructions

```
Sub FunctionWithErrors()
 MsgBox ("Confirm receipt of message")
 James Tam
 Tutorial 1
End Sub
```

Instructions for the computer

*(Computer): problem, I don't know how to "James Tam"*

## Program Documentation (2)

- You must 'mark' this information so it doesn't cause an error
  - The marking will indicate to the VBA translation mechanism that the line is for the reader of the program and not to be translated and executed
  - The marking is done with the single quote '
- **Format:**
  - ' *<Documentation>*
- **Example:**
  - ' **Author: James Tam**
- No error: Everything after the quote until the end of the line will not be translated into machine language/binary
- That means documentation doesn't have to be a valid and executable instruction

## Program Documentation (3)

- Contact information should be located before your program
- Before the 'sub' keyword

```
' Author: Smiley
' Student ID: 123456
' Tutorial 01
Sub FunctionWithErrors()
 MsgBox ("Confirm receipt of message")

```

} Documentation:  
marked in red

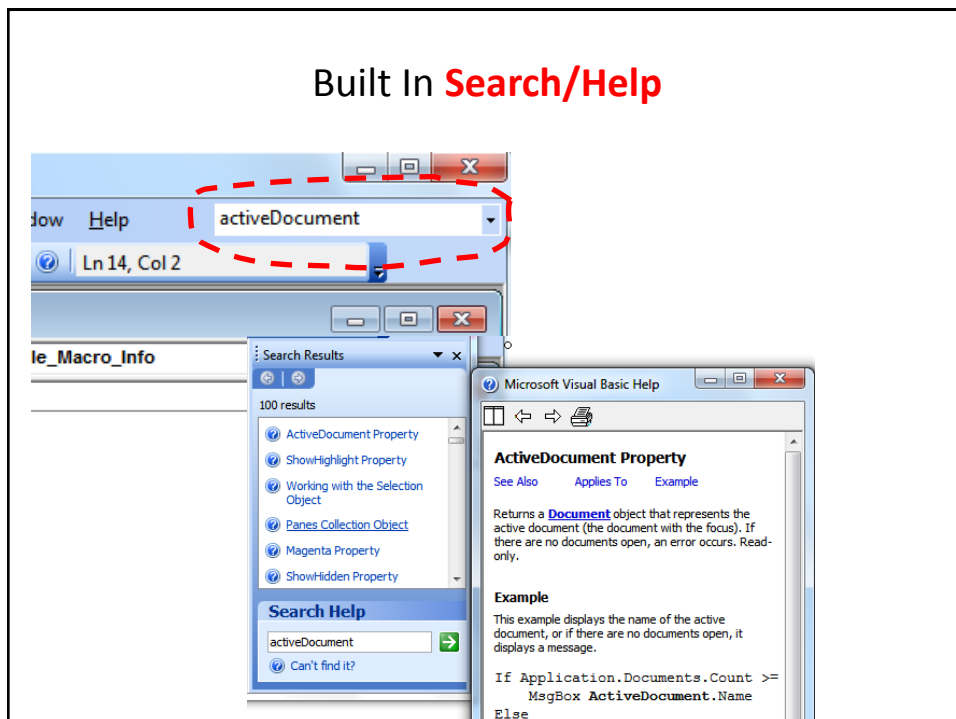
## Program Documentation (4)

- Program features (this will be worth many marks)
- Assignment description

1. Word search and replace: The user is prompted for the word to search for and the word to replace it with. If the word to search for does not exist in the document then nothing will happen.
2. Style search and replace: The program will search and replace for all instances of the style "Heading 1" within the document.
3. Alphabetically sort table the contents of a table in ascending order ('a' to 'z'); you will get credit for implementing the program to sort the contents *within each table but not between tables* (e.g., the first entry of the second table can come before the first entry of the first table).
  - Basic version: Your program is written so it can sort a fixed number of tables only e.g., my sample document has 3 tables.
  - Advanced version: Your program can sort a document with any number of tables. If there are no tables in the document, the program should prompt the user.
4. Run a spell check of the document.
5. Print the document (you won't actually be able to print anything in the 203 labs because there are no printer drivers installed on the machines, but you can use the Print command using VBA).
6. Save the document (your program doesn't have to check if any changes were actually made to the document before saving).
7. Close the document.

- Program documentation
  - ' Author: James Tam ID: 123456
  - ' Version: Nov 2, 2015
  - ' Tutorial: 99
  - ' PROGRAM FEATURES
  - ' Word search and replace: completed
  - ' Style search and replace: not completed
  - etc

## Built In Search/Help



## Another Source Of Help

- The macro recorder!
- Although you are writing your programs manually (typing them in) you can use the macro recorder to help you determine which objects, methods and properties are needed for a particular task.
- Example (from toggling bold and italic font):
 

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

## Recorded Macros

- Sometimes the auto generated code is too complicated to be useful
  - Toggle bold, italic, printed document

```
Sub Macro3()
 Selection.Font.Bold = wdToggle
 Selection.Font.Italic = wdToggle
 Application.PrintOut FileName="",
 Range:=wdPrintAllDocument, Item:= _
 wdPrintDocumentWithMarkup, Copies:=1, Pages="",
 PageType:= _
 wdPrintAllPages, Collate:=True, Background:=True,
 PrintToFile:=False, _
 PrintZoomColumn:=0, PrintZoomRow:=0,
 PrintZoomPaperWidth:=0, _
 PrintZoomPaperHeight:=0
End Sub
```

## After This Section You Should Know

- The history and background behind VBA
- How to copy and run the pre-created lecture examples
- How to create and execute simple VBA macros
  - Automatically recording macros
  - Manually entering programs into the VB editor yourself
- How to create/use a Message Box “MsgBox”
- How the VB editor identifies programming errors
- What is a VB object, how to use the properties and methods of objects
- How to use basic mathematical operators in VB expressions
- How to create and use variables
- How to use the title bar to display information

## After This Section You Should Know (2)

- What is a named constant, why use them (benefits)
- What is a predefined constant and what are some useful, commonly used predefined constants
- Naming conventions for variables and constants
- What are commonly used variable ‘types’ in VB
- How to get user input with an Input Box “InputBox”
- How/why use the VB debugger
- Common formatting effects that can be applied to an entire document or selected parts
- How to create program documentation (as well contact information that should be included in documentation)

### After This Section You Should Now Know (3)

- The security settings in the MS-Office “Trust Center”
- How different types of MS-Word documents have different levels of security
- How to print the currently active Word document using a macro