

## 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

## B.A.S.I.C.

- Beginner's All-Purpose Symbolic Instruction Code (BASIC)
  - From: [www.acm.org](http://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.
- However 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

## 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](mailto:tamj@ucalgary.ca) with another [tamj@cpsc.ucalgary.ca](mailto: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 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.

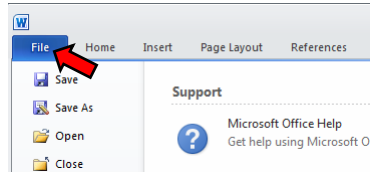
## Advanced Use Of Macros (2)

- 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

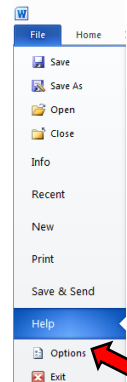
## Viewing The 'Developer' Ribbon (MS-Word 2010)

- 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



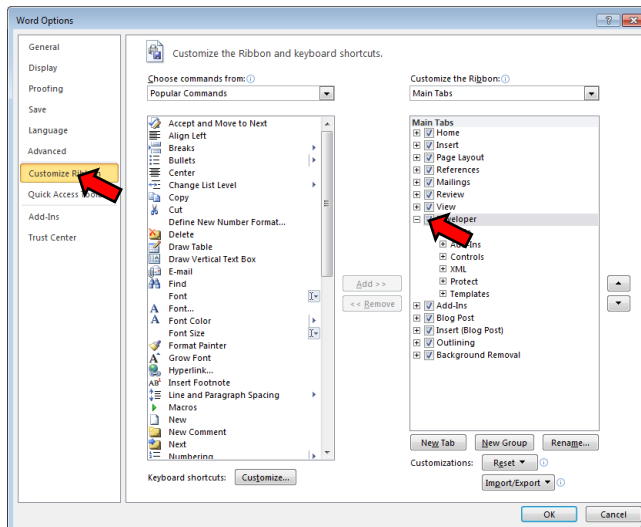
2. Select 'options'



## Viewing The 'Developer' Ribbon (MS-Word 2010): 2

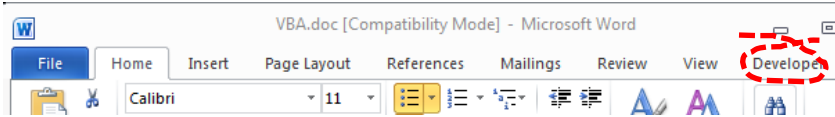
3A) Select "customize the ribbon"

3B) Check the 'Developers' box



## Viewing The 'Developer' Ribbon (MS-Word 2010): 3

- This should add a new ribbon "Developer"



## 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!

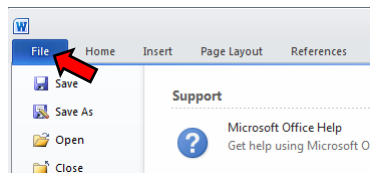
## Examples Macros Viruses

- “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.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](http://ca.norton.com/search?site=nrt_n_en_CA&client=norton&q=macro+virus)

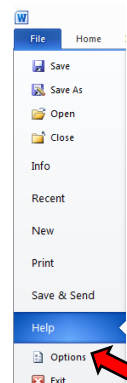
## Enabling Macros To Run

- 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'

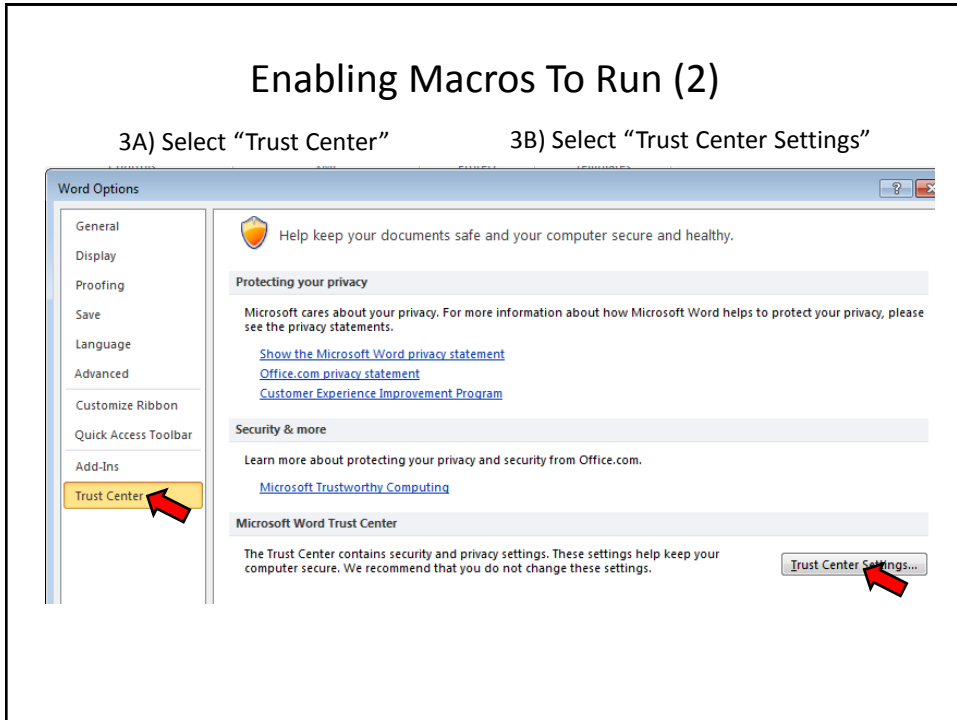




## Enabling Macros To Run (2)

3A) Select "Trust Center"

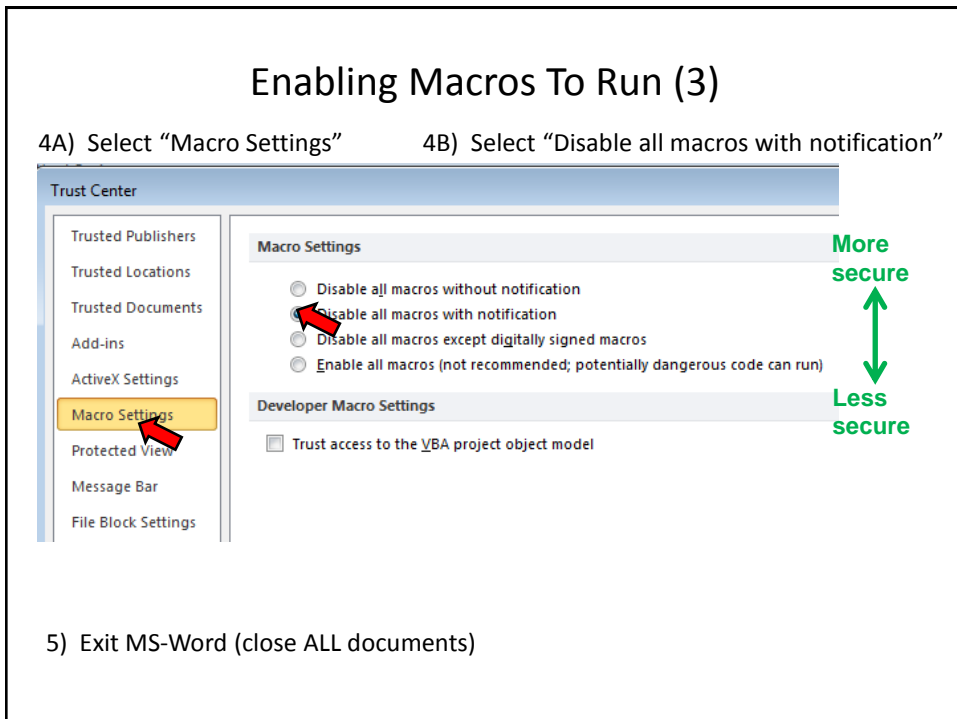
3B) Select "Trust Center Settings"



## Enabling Macros To Run (3)

4A) Select "Macro Settings"

4B) Select "Disable all macros with notification"



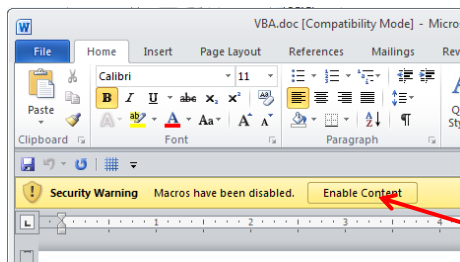
5) Exit MS-Word (close ALL documents)

## Enabling Macros To Run (4)

- This should be the default setting for MS-Word (“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.

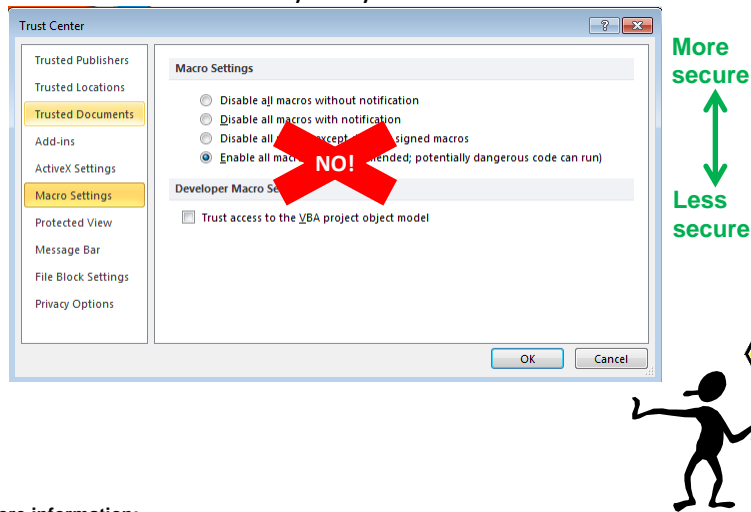


### JT's caution

- You should not casually select this option for all MS-Word documents
- It's recommended that you ONLY do it for ones you have created.

## Macro Security

- DO NOT take the 'easy' way out



For more information:

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

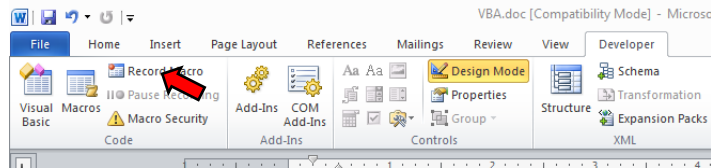
## Creating Macros

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)

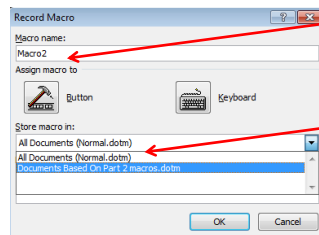
## Recording Macros

- Developer ribbon

– “Record Macro”



– Recording details



What to  
name the  
macro

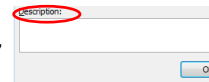
Where to  
store the  
macro

## 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 e.g., 'formatting\_resume\_headings'

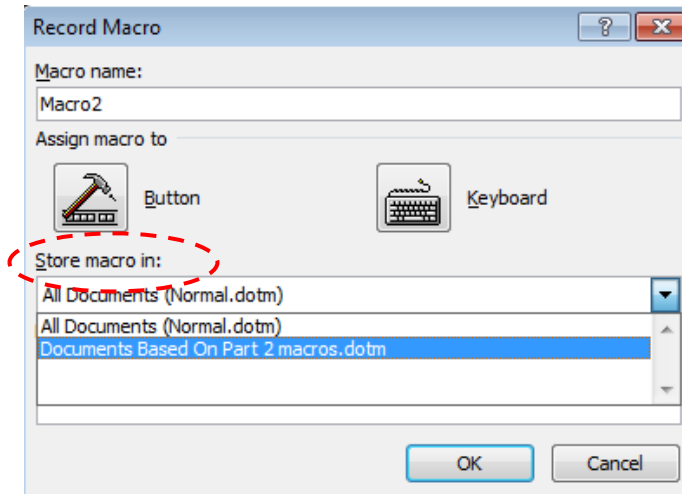
– Additional details can be provided under 'description' (more later)



- Language requirements:

- Must start with an alphabetic letter, after than any combination of letters and numbers may be used
- 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)

## Where To **Store**

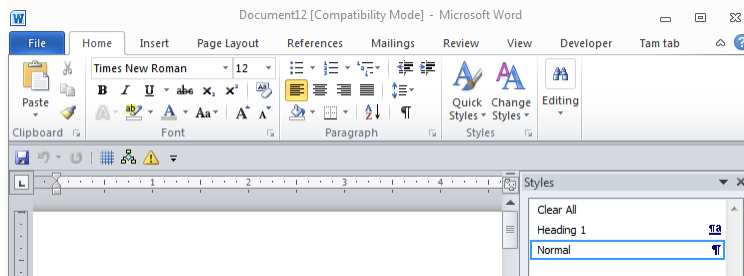


### Side Note: Template Documents

- Can be used to customize MS-Word settings e.g., creating formatting styles
- You can have the customizations will apply each time that you open MS-Word.
- If you want different customizations to apply under different circumstances then you can create template documents.

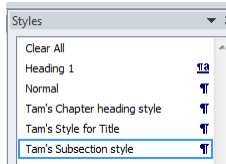
## Side Note: Template Documents (2)

- Example: default formatting styles



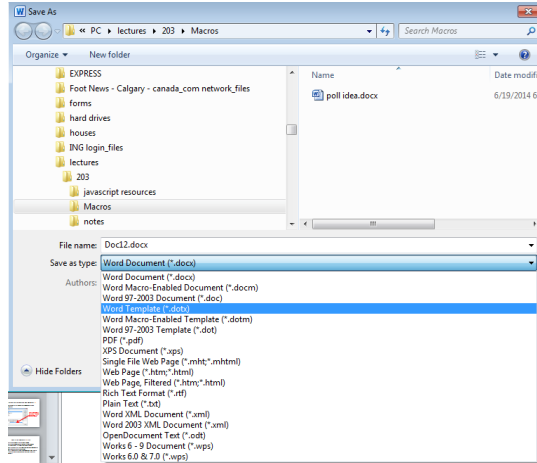
## Side Note: Template Documents (3)

- Example: you can create custom formatting styles



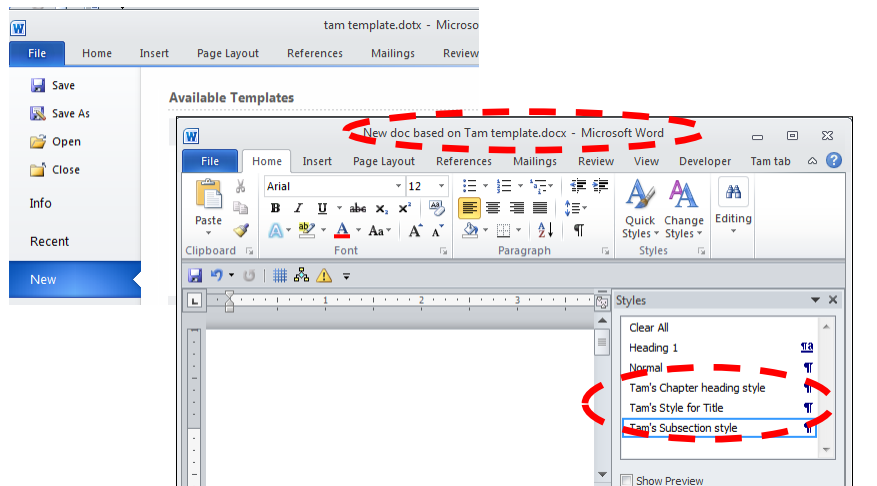
## Side Note: Template Documents (4)

- You have these customizations available only under certain situations by creating a template document ('.dotx')



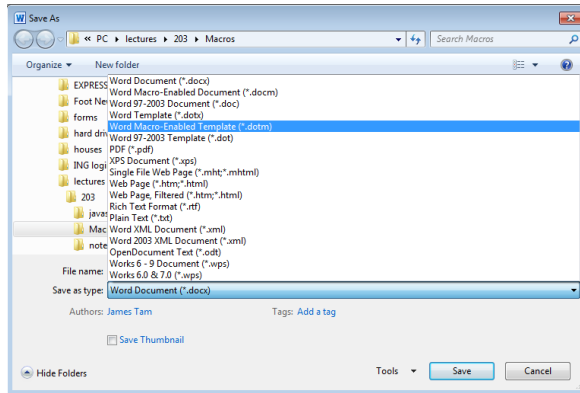
## Side Note: Template Documents (5)

- All documents created using this template will have access to the customizations.



## Document Templates & Macros

- You can also create templates that not only include formatting customizations but also include macros that you create.
- In the latter case: The template must be saved as a **macro-enabled template** rather than a regular template (‘.dotm’)



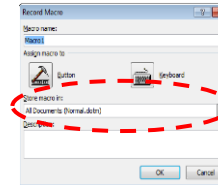
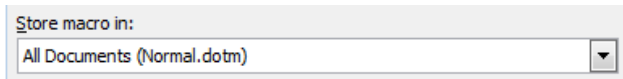
## Storage Locations For Macros

1. All templates: all MS-Word documents can access the macro
  - Located in the 'normal.dotm' document
2. User created template: only MS-Word documents based on that template can access the macro
  - Creating your own '.dotm' template document
3. One document: the macro instructions can only be accessed in the one document
  - Creating a macro enabled document '.docm'



## #1 Making Macros Available In *All Documents*

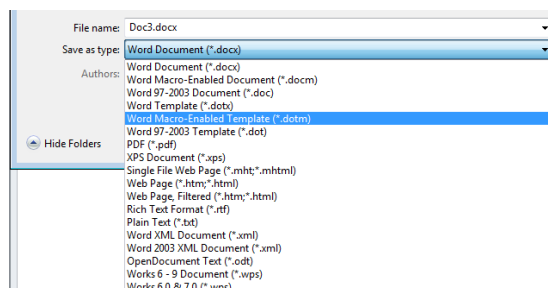
- This is the default option for saving macros



- (Note: this is the most convenient approach if you work on the same computer.
  - This approach is not recommended if you work on multiple machines (e.g., 203 lab and computer(s) at home) because transferring your work will take more work.
  - If you do employ this approach on your own computer then you will probably want to delete all your macros associated with the 'normal' template after you finish with this course (details on 'deleting macros' coming up).
  - For these reasons this approach is **NOT recommended** for this course.

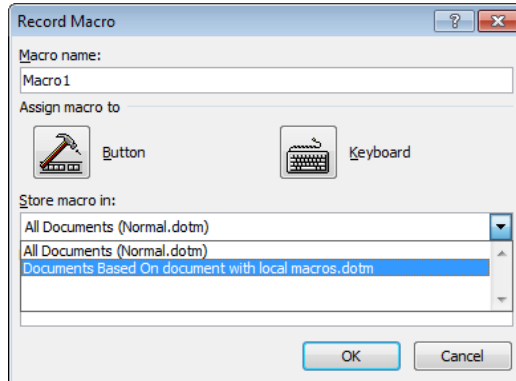
## #2: Making Macros Specific To *One Template*

- In 'real life' your organization may be creating so many macros in documents that some macros will only be used by some users.
- Macros can be organized into different template 'doc-dotm' documents (*m = macro enabled document*)
- Steps:
  - Create a new macro-enabled template



## #2: Making Macros Specific To One Template (2)

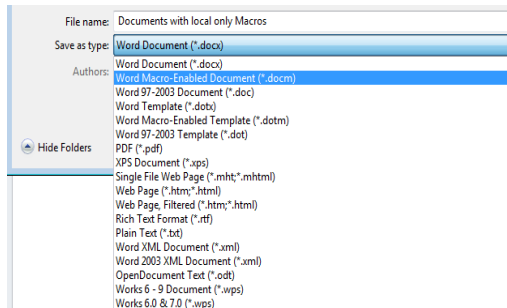
- After creating a new template you can now make your new macros specific to that new template:



- You can transfer your work from one computer to another by transferring the template document (**the recommended approach for this class**)

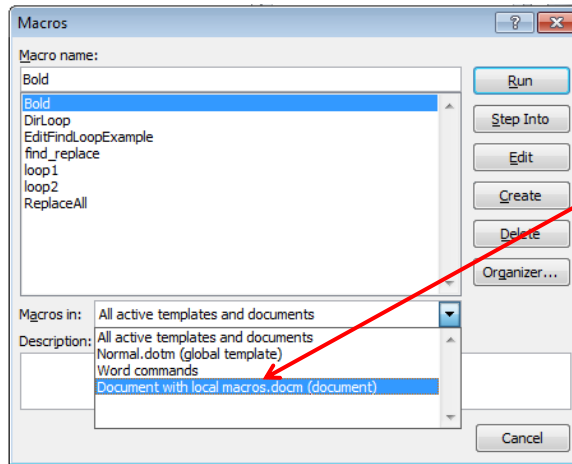
## #3: Making Macros Specific To *One Document*

- Save the document as one that has macros-enabled via “save as” rather than just “save”



### #3: Making Macros Specific To One Document (2)

- Recorded macros can be made available in only this one document



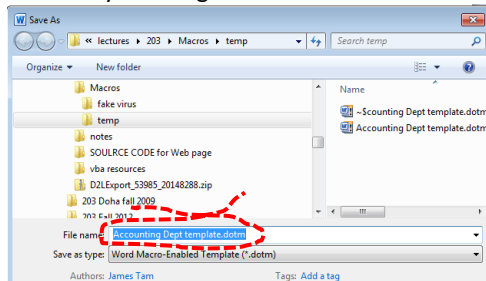
This is the name of the document "Document with local macros.docm"

### Contrasting Approach #2 & #3

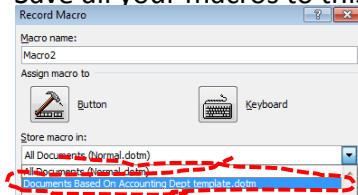
- #2 Creating a new macro-enabled template
- #3 Creating a new document with macros enabled in it
- Both seem similar (creating a file using MS-Word)
- The difference is that the second approach only allows macros to work in *one* MS-Word document.
- The first approach creates a template with macros that can work in *all* MS-Word documents that are *based on that template*.
- One of way of transferring macros between machines is to transfer the macro-enabled document or macro-enabled template.

## Saving Macros

- Simple recap: If you don't know where to save your macros:
  - Start by creating a new macro enabled template

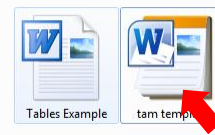
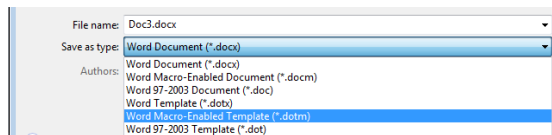


- Save all your macros to this template ('.dotm')

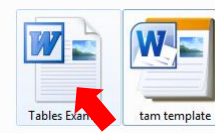
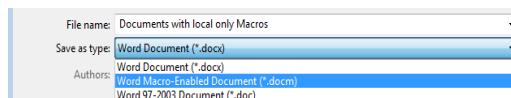


## Transferring Your Macros (This Class)

- If you created a macro-enabled template then transfer the template document (again file name suffix ".dotm")

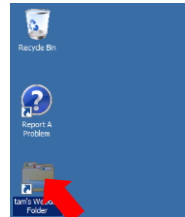
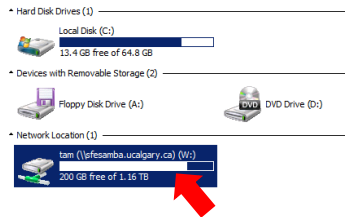


- If you create a macro-enabled MS-Word document (file name suffix ".docm") then transfer the Word document itself.



## Transferring Your Macros (This Class): 2

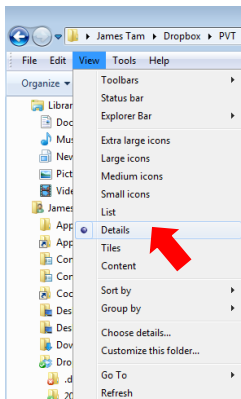
- In the 203 computer lab
- Save the template document or macro enabled word document to your portable USB flash drive
- OR
- Save the template document on your web disk drive



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

## Viewing File Information

- View details: select 'view' in a folder

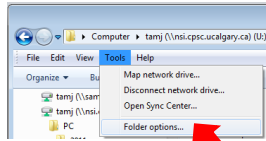


Effect

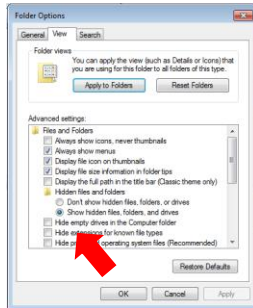
|                |                   |                                   |
|----------------|-------------------|-----------------------------------|
| Tables Example | 8/15/2014 5:34 PM | Microsoft Word 97 - 2003 Document |
| tam template   | 8/26/2014 4:59 PM | Microsoft Word Template           |

## View File Suffixes

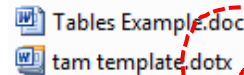
- In a folder select: Tools->Folder options



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



Effect

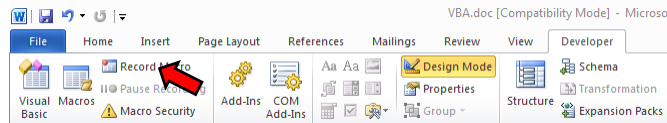


## The First Simple Macro

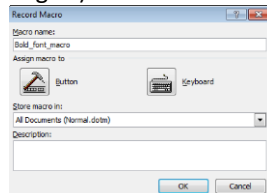
- With word processing there's sometimes a need to apply multiple formatting styles (bold, italics, underline) to highlighted text
- Manually applying the required formatting to each block of text can be tedious
  - Recall: Macros can be used to automate or shorten some tasks
- This first example macro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

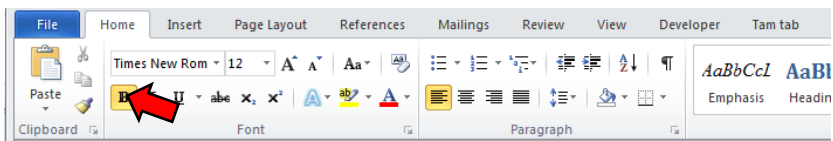


- Give the macro a self explanatory name and press 'OK' (recording begins)

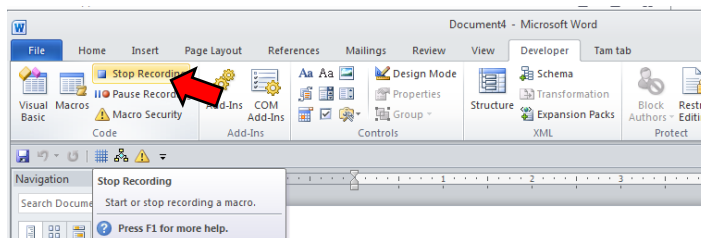


## Recording A Macro (2)

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

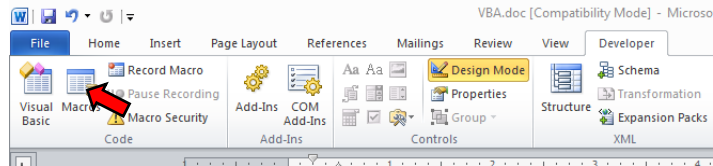


- All commands have been entered so you can stop the recording

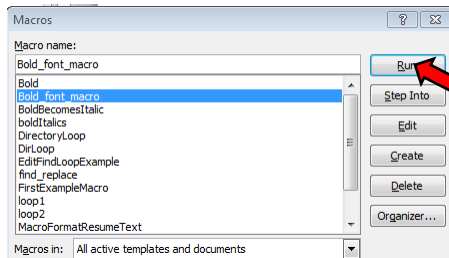


## Running A Recorded Macro

- Under the Developer ribbon select 'macros'

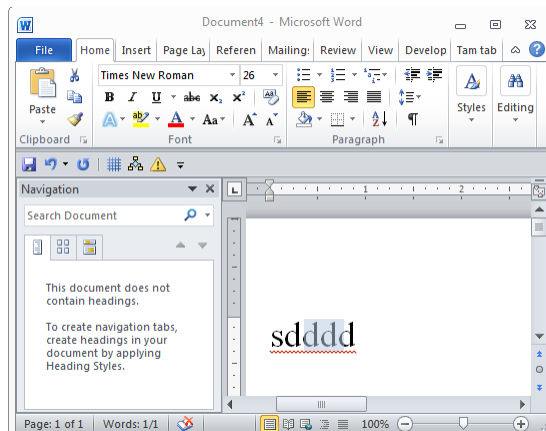


- Select the macro and then 'run' it



## Running A Recorded Macro (2)

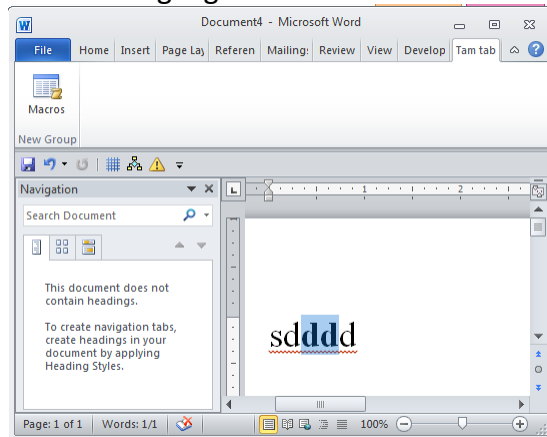
- In this case nothing happened?
  - This macro changes selected/highlighted text to bold
  - You need to select some text before running the macro





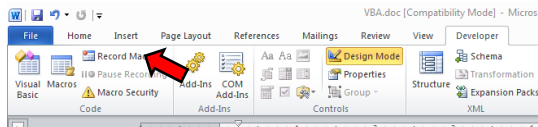
## Running A Recorded Macro (3)

- 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

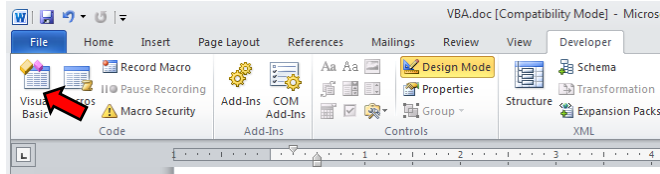
```
Sub AutoGeneratedFontChange()  
    Selection.Font.Bold = wdToggle  
End Sub
```

## Recording Macros

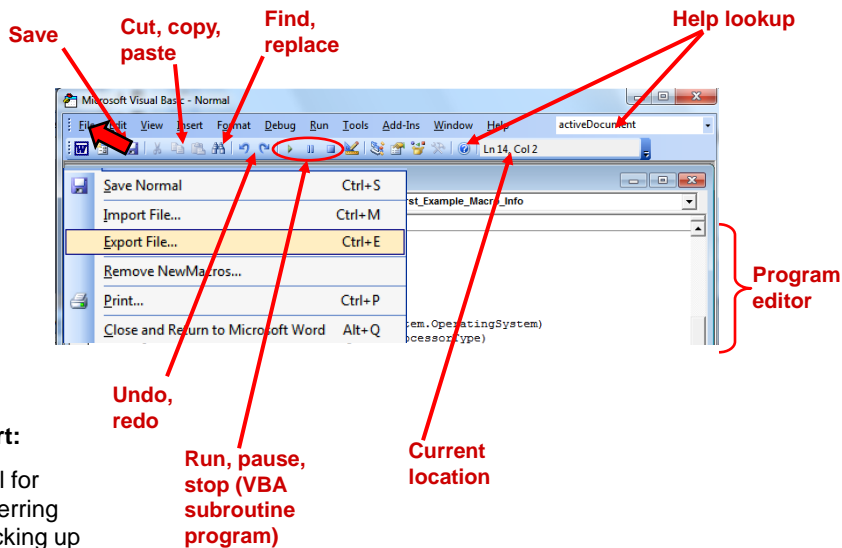
- Bottom line: use it for learning “how to do” things
- Don't:
  - Just use the auto-generated code to study for the exam without creating any code of your code
  - Just hand in the auto-generated VBA code for your assignment
- Use the auto-generated code to figure out how to “type the program from scratch” yourself (I will show how to do this shortly)

## 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



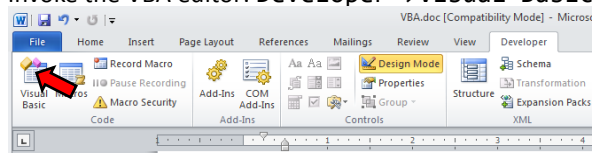
**Export:**  
Useful for  
transferring  
or backing up  
your work

## VBA Examples: This Point Onwards

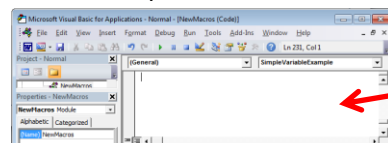
- Unlike the previous example you will be manually typing in the program instructions yourself rather than automatically recording the program as a series of steps
- Reminder: some of the early examples are meant only as a learning/teaching tool
  - They show you how to write simple VBA programs
  - 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

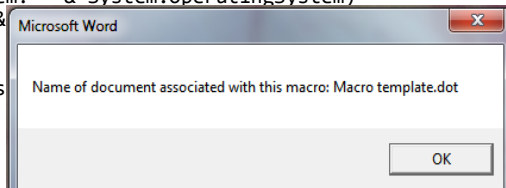


Enter your program instructions here (program editor)

## First VBA Example (2)

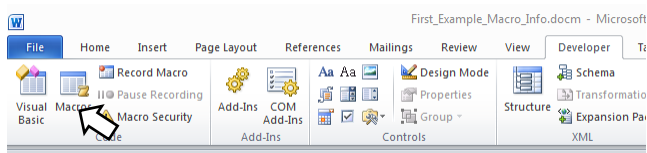
- Type in or cut-and-paste the following example into the *VBA editor* (see last image for location of the editor, previous slide)
  - This is **NOT** the same as pasting it directly into MS-Word.
  - **Word document containing the macro:**  
first\_example\_macro\_info.docm

```
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.Processor)
' Document information
MsgBox ("Name of document associated with this macro: " & ActiveDocument.Name)
End Sub
```

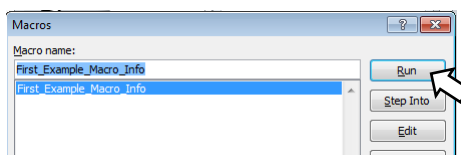


## Reminder: Running Macros

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

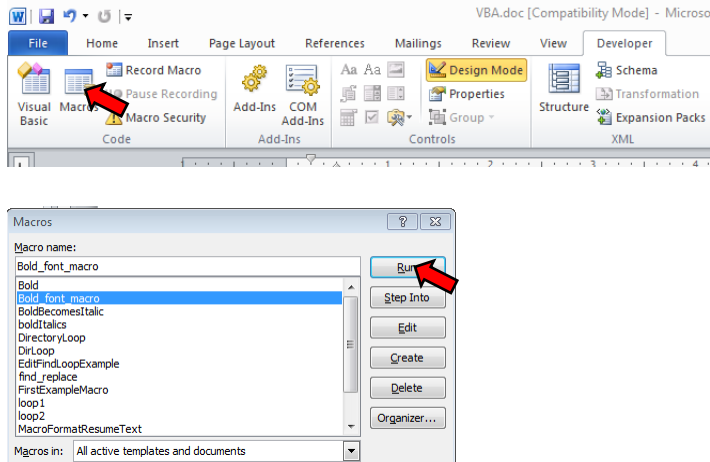


- The single macro should be highlighted, then click 'run'



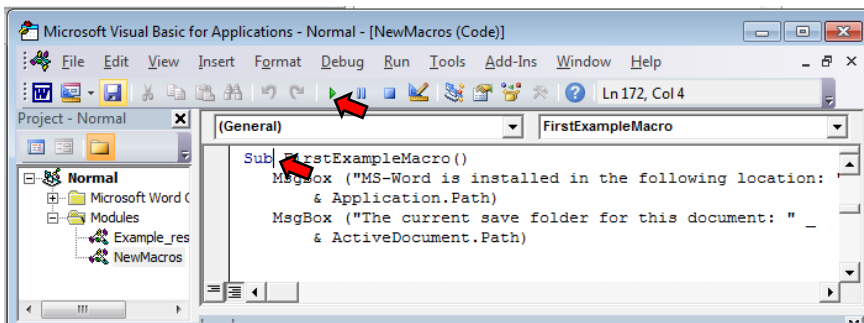
## Running VBA Programs You Have Entered

- It can be done the same way as with auto-recorded programs



## Running VBA Programs You Have Entered (2)

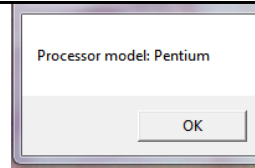
- Or you can run the program right after you have entered it (in the editor).



1. Ensure correct program "sub" is to be executed (click there)

2. Press the 'play/run' button or "F5"

## 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>"*)

**Ampersand '&':**  
connects strings (with quotes) and function return values (no quotes)

- **Example:**

**MsgBox** ("Core operating system: " &

**Quotes: literal string**  
appears in message box

System.OperatingSystem)

**No quotes: the return**  
**result of function (this**  
**example) or contents of**  
**variable or constant**  
**(later examples)**

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)



## Early Exam Information

- You should know how a message box works and how to create one
- Unless a function is extremely common or you are told otherwise you won't have to memorize how a specific function operates e.g., `System.OperatingSystem` (too specific for the exam).
- Alternatively you may be given a reminder in the exam itself that describes how to use a particular function.
  - In this case you should then be able to write statements that employ that function.

## VBA Visual Aids: **Function Arguments**

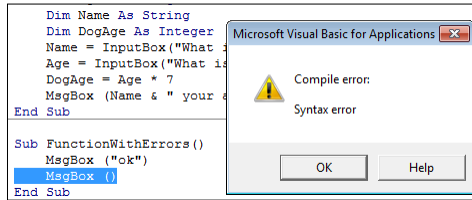
- As you type in the name of VB functions you will see visual hints about the arguments/inputs for the function.
  - Enter the function name and then a space

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

**Function arguments**

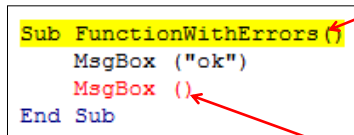
**(Bold): mandatory arguments**

## VBA Visual Aids: **Error Information**



**Required argument missing**

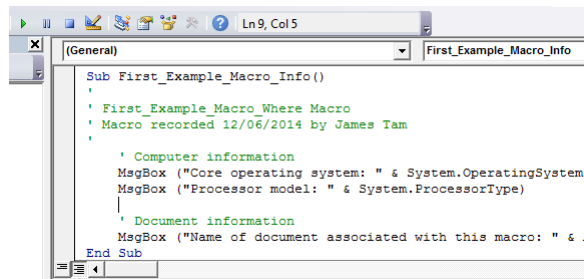
**Part of program that contains errors (yellow highlight)**



**Specific statement/instruction causing the error (red font)**

## Viewing Macros

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



**Previous example: auto recorded**

## 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)**

#### End of subroutine:

- Has 'End Sub'

- **Example:**

```
Sub First_Example_Macro_Info()
```

```
End Sub
```

- All executable VBA program statements must be inside the subroutine

## Objects

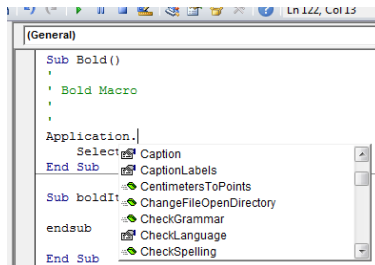
- You are of course familiar with objects in the everyday world.
  - These are physical entities



- Similar to everyday objects MS-Objects have properties and capabilities
  - 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

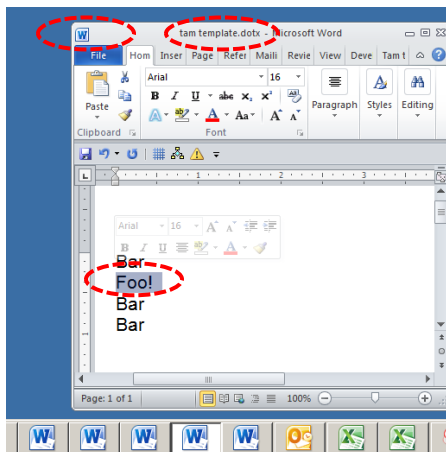
- **Application**: the MS-Office program running (for CPSC 203 it will always be MS-Word)
- **ActiveDocument**
- **Selection**
- When enter one of these keywords in the editor followed by the 'dot' you can see more information.



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!"*

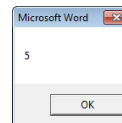
## Using Pre-Built Capabilities/Properties of Objects

- **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'

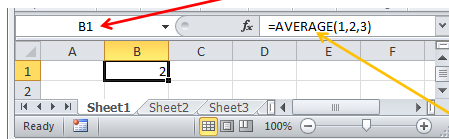
Accessing the Windows property of  
Word (the application)  
• Info about the windows currently opened

## Properties Vs. Methods/Functions

- Recall

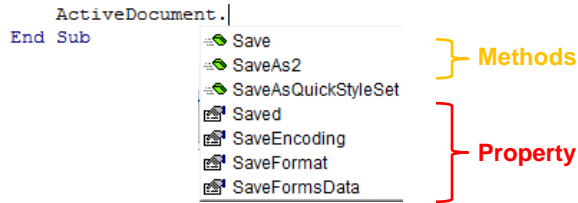
- **Property:** information about an object
- **Method:** capabilities of an object (possible actions)

Property:  
current cell



Using the  
'average()' function

## Properties Vs. Methods: Appearance

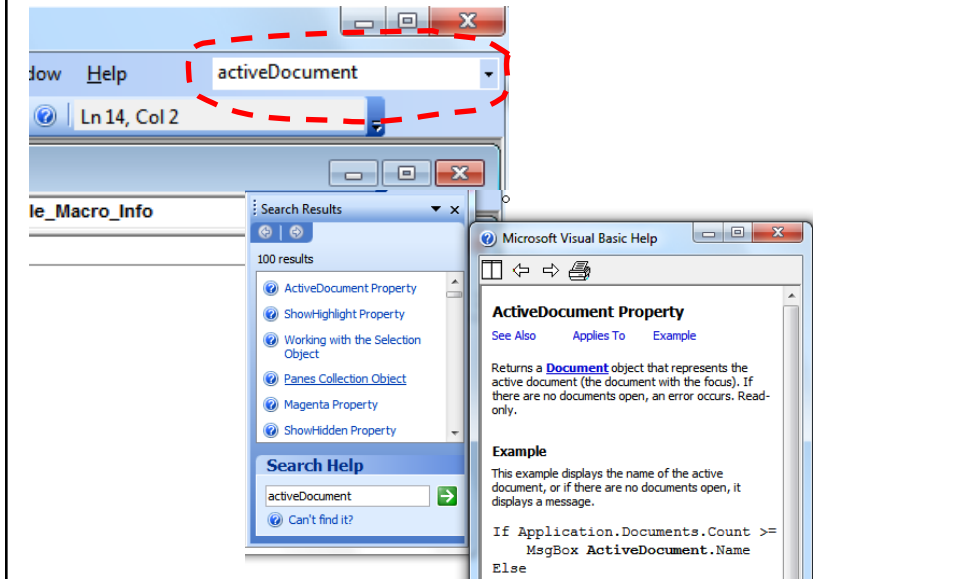


- Similar to functions in MS-Excel some object's methods may require an argument or arguments
- Examples
  - `ActiveDocument.CountNumberedItems` ← No argument required
  - `ActiveDocument.Save` ← No argument required
  - `ActiveDocument.SaveAs2("<name>")` ← Argument: New name of document needed

## 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)

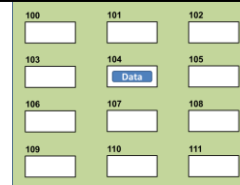
## Built In Search/Help



## Basic Mathematical Operators

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

## 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 <sup>1</sup>    | String <sup>2</sup> | Dim Name As String           | Empty string  |
| Date <sup>3</sup>          | 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"
```

## Second VBA Example

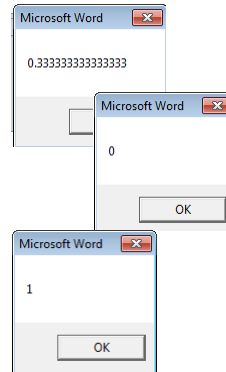
- Learning Objectives:
  - Using variables
  - Using mathematical operators

## Second VBA Example (2)

**Word document containing the macro:** secondExample.docm

```
Sub FirstExample()
    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**

## 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).

## Variable Naming Conventions (2)

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

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

## Variable Naming Conventions (3)

- |   | <b>Examples</b>  |
|---|--|
| 3. Style requirement: Variable names should generally be all lower case except perhaps for the first letter (see next point for the exception).   | #3:<br>Age, height, weight (yes)<br>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<br>firstName, last_name<br>(yes to either approach) |
| 5. Avoid using keywords as names (next slide)   |  |

## Some Common Visual Basic Keywords<sup>1</sup>

|          |           |         |           |          |          |
|----------|-----------|---------|-----------|----------|----------|
| 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      |         |           |          |          |

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

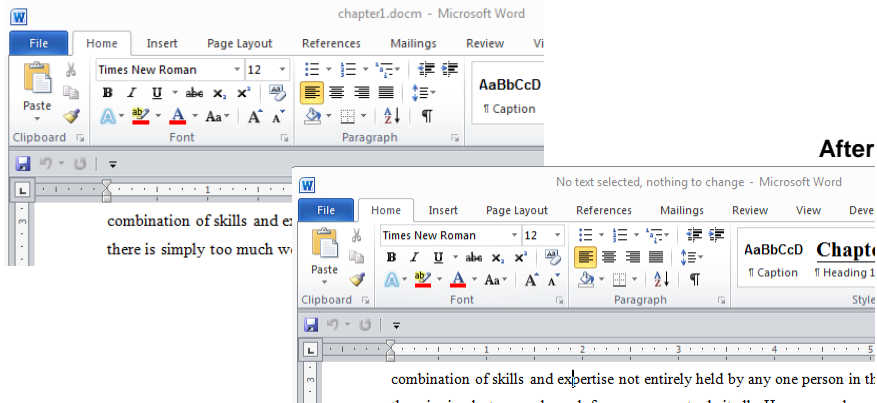
## Variable Naming Conventions: Bottom Line

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

## An Alternate To The MsgBox: The Status Bar

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

### Before



## Changing The Title Bar

- **Format:**

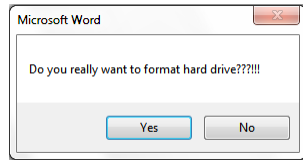
```
ActiveDocument.ActiveWindow.Caption =
    "<New text for the title bar>"
```

- **Word document containing the macro:**  
firstTitlebarExample.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

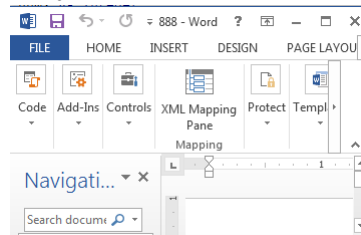
- **Status messages:**
  - More subtle presentation of information.
  - Easy to miss.
  - Multiple updates result in only the last change appearing to the user

## Missing The Message

**Word document containing the macro:**  
secondTitleBarExample.docm

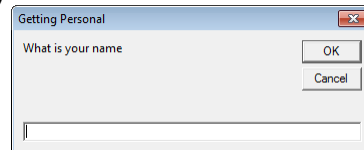
```
Sub secondTitleBarExample()
    Dim num1 As Integer
    Dim num2 As Integer

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



## Getting **User Input**

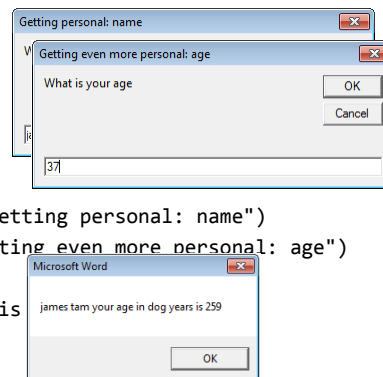
- A simple approach is to use an **Input Box**
- Format:  
`<Variable name> = InputBox(<"Prompt">, <"Title bar">)`
- Example:  
`Name = InputBox("What is your name"), "Getting Personal")`
- Note: only the string for the prompt is mandatory.
- If the title bar information is omitted then the default is the application name ("Microsoft Word")



## Third Example: InputBox

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

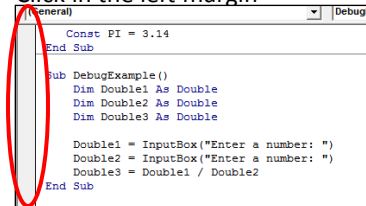
```
Sub InputExample()
    Dim Age As Integer
    Dim Name As String
    Dim DogAge As Integer
    Name = InputBox("What is your name", "Getting personal: name")
    Age = InputBox("What is your age", "Getting even more personal: age")
    DogAge = Age * 7
    MsgBox (Name & " your age in dog years is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



```

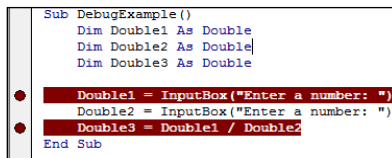
General
Const PI = 3.14
End Sub

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
  
```

## The VBA Debugger (2)

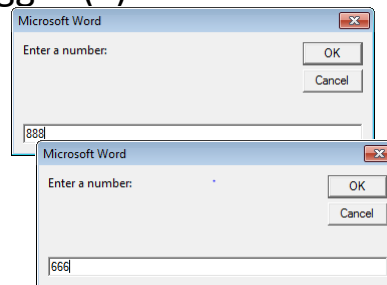
- Multiple breakpoints



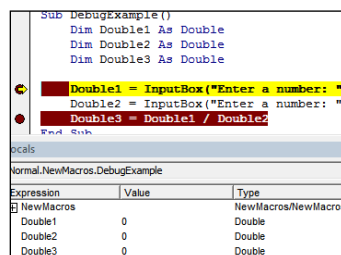
```

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

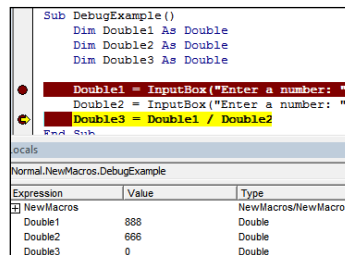


```

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  |       | 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  |       | NewMacros/NewMacros |
| Double1    | 888   | Double              |
| Double2    | 666   | Double              |
| Double3    | 0     | Double              |

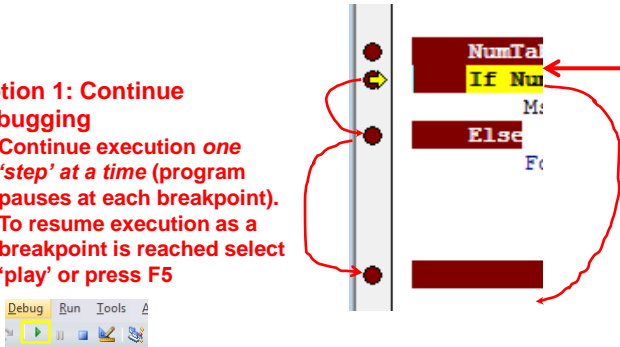


## How To Stop Step-By-Step Execution: Step Out

- Suppose you have found and fixed the error and want execution to resume normally (without stopping at each break point)
- You can “step out” of the breakpoints.
- Example

**Option 1: Continue debugging**

- Continue execution *one 'step' at a time* (program pauses at each breakpoint).
- To resume execution as a breakpoint is reached select 'play' or press F5



**Option 2: Stop debugging**

- *Step out of debugging* by pressing F8
- Program executes without stopping

Suppose execution reaches this point

NumTal  
If Num  
M:  
Else  
F

Debug Run Tools A

## 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

## 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>**<sup>1</sup>

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

- **Example:**  
Sub ConstantExample()  
    **Const PI = 3.14**  
End Sub

<sup>1</sup> 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: wdFormatOriginalFormatting

## Predefined Constants

- They are constants that are already defined by the language.
- Similar to named constants that you create their use can make your program easier to read and understand.
- **Word document containing the macro: cutAndPaste.docm**
  - Learning concepts: Cutting and pasting text, using constants

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

The *cat* in the hat.      The ~~cat~~ in the hat.

The *cat* in the hat.

The *cat cat* in the hat.

- A predefined constant
- Another is "wdFormatPlainText"

PasteAndFormat(1)
 

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

## Student Exercise (Early Review Example)

- What would happen to the following document...

The *cat in the hat*.

- ...if the following macro was executed?

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



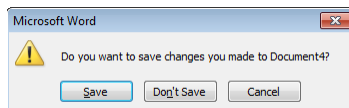
JT: This is a timed in class exercise (you get the answer in lecture so be sure to come to class)

## Previous Example: For The Exam

- Without any additional documentation or explanation you should know
- The Selection Object:
  - Expand()
  - Copy()                    ‘ Also you should know Cut()
  - MoveRight()            ‘ Also you should know MoveLeft()
  - PasteAndFormat()    ‘ Also you should be familiar with predefined constants:  
                                   wdFormatOriginalFormatting  
                                   wdFormatPlainText

## 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:**  
"ClosingActions.docm"

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

'Choose one constant  
**wdPromptToSaveChanges**  
**wdDoNotSaveChanges**  
**wdSaveChanges**



```
End Sub
```

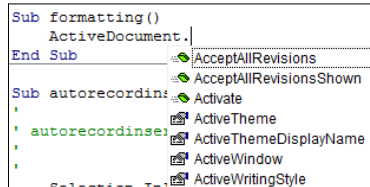
## Formatting Text

- Examples: bold, underline, font type, font size
- Formatting effects can be applied to an entire document
  - Used when the formatting effects apply to the entire document
- Formatting effects can be applied only apply to the currently selected part 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
```



- 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.

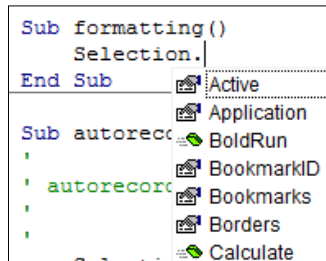


- So it's best to work with only a single document at a time when writing your programs.

## Formatting The Text Currently Selected

- This can be done through the 'Selection' object

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



The screenshot shows a VBA editor window with a code editor on the left and a Properties window on the right. The code editor contains the following code:

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

The Properties window on the right shows the 'Selection' object with the following properties and methods:

- Active
- Application
- BoldRun
- BookmarkID
- Bookmarks
- Borders
- Calculate

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

## Formatting Text: An Example

- Suppose you want to format a document in the following way
- Entire document
  - Font = Calibri
- Selected headings
  - Font size = 14 point
  - Bold
  - Underline
  - Centre the heading
  - 6 point spacing after the heading
- Selected paragraphs
  - Font size = 10 point
  - Left and right margins indented by 0.2 inches



## 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:**  
formattingEntireDocument.docm

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

## Formatting Selected Headings

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

```
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 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.

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?

## Program Documentation

- Your VBA assignment submission must include identification information:
  - Full name
  - Student identification number
  - Tutorial number
- DON'T just enter this information into your program instructions

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

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

Instructions for the computer

## 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

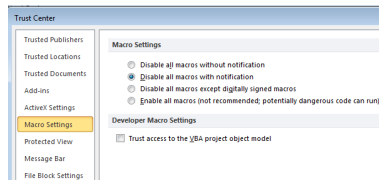
```
' Author: Smiley
' Student ID: 123456
' Tutorial 01
Sub FunctionWithErrors()
    MsgBox ("Confirm receipt of message")
End Sub
```

Documentation:  
marked in red

- (More on program documentation will come later).

## More On Security

- You should know
  - Details about macros
    - 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 setting in MS-Office ("Trust Centre")






## More On Security (2)

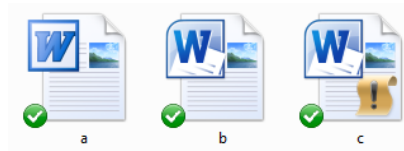
- You should know (continued)
  - Where macros can be 'stored'
    - Normal 'dot-m' template
    - Custom 'dot-m' template
    - In a single document 'doc-m' document

## Question: What Is The Security Difference?

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


## How Can You Determine The Document Type?


|   |                  |                  |
|---|------------------|------------------|
|  a | 9/2/2014 1:17 PM | Microsoft Word!  |
|  b | 9/2/2014 1:17 PM | Microsoft Word I |
|  c | 9/2/2014 1:18 PM | Microsoft Word I |



## .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

## 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