Excel Macros and VBA

CEE3804 Computer Applications for Civil and Environmental Engineers

Topics to be Covered

- Excel Macros
- Understanding and making use of VBA
- Basics of VBA
    - Using code modules
    - Understanding procedures
    - Interacting with the user
- Creating useful forms
    - Adjusting form layout
    - Using form and control events

Macros
Definition

- A macro is:
  - a series of commands recorded within the user interface and wrapped into a single action
- A procedure is:
  - is a series of actions but, unlike macros, a procedure is written from scratch with the Visual Basic for Applications (VBA) programming language
- In summary:
  - a series of commands is called a macro when it is recorded, however, a macro is a procedure within the VBA world

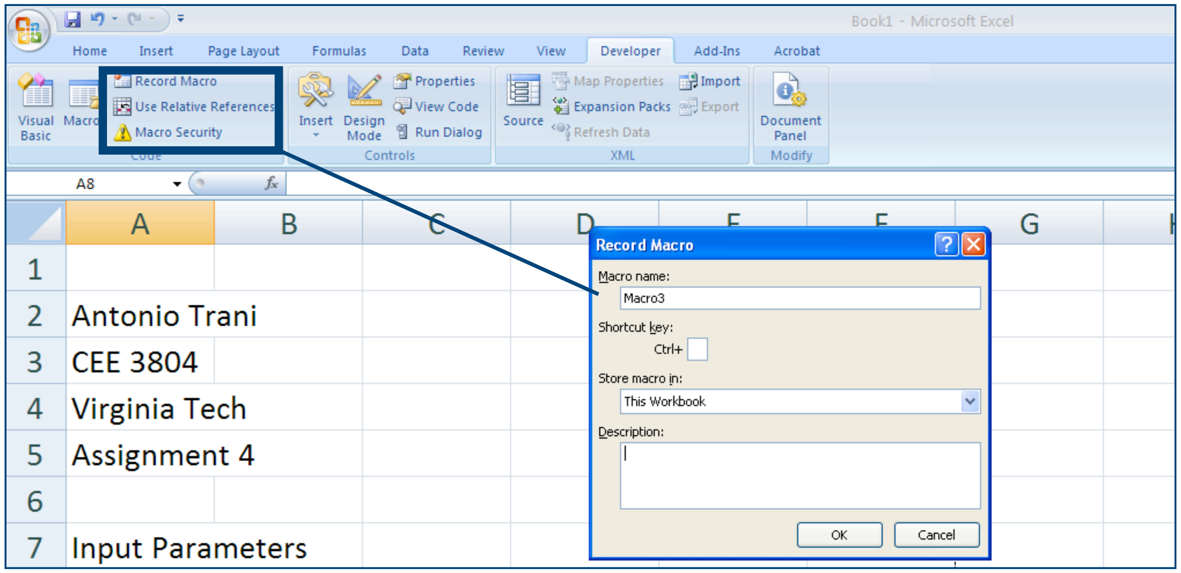
Macros
Why use Macros?

- Why use macros?:
  - to simplify a series of commands by automating the task
  - simplify complex tasks
  - to learn how the VBA language lends itself to the Excel environment

Macros
Recording Macros

- Recording macros:
    - Tools/Macros/Record New Macro
    - Macro recorder is impartial:
        - should map out exactly what you are trying to do
            - overall goal of macro
            - cells that will be selected
            - data required by macro
            - menu command to accomplish task
            - workbooks that will use the macro
    - Give macro a descriptive name and shortcut
    - Indicate relative versus absolute references

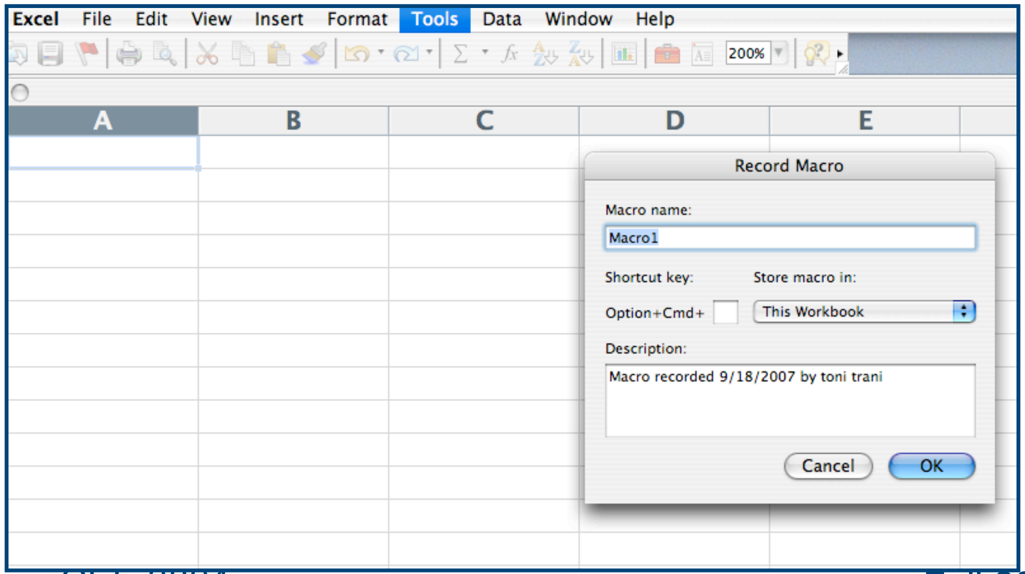
Macros : Recording



	A	B	C	D
1				
2	Antonio Trani			
3	CEE 3804			
4	Virginia Tech			
5	Assignment 4			
6				
7	Input Parameters			

Record Macro
Macro name: Macro3
Shortcut key: Ctrl+
Store macro in: This Workbook
Description:
OK Cancel

Excel 2007
Look for the
Developer Tab

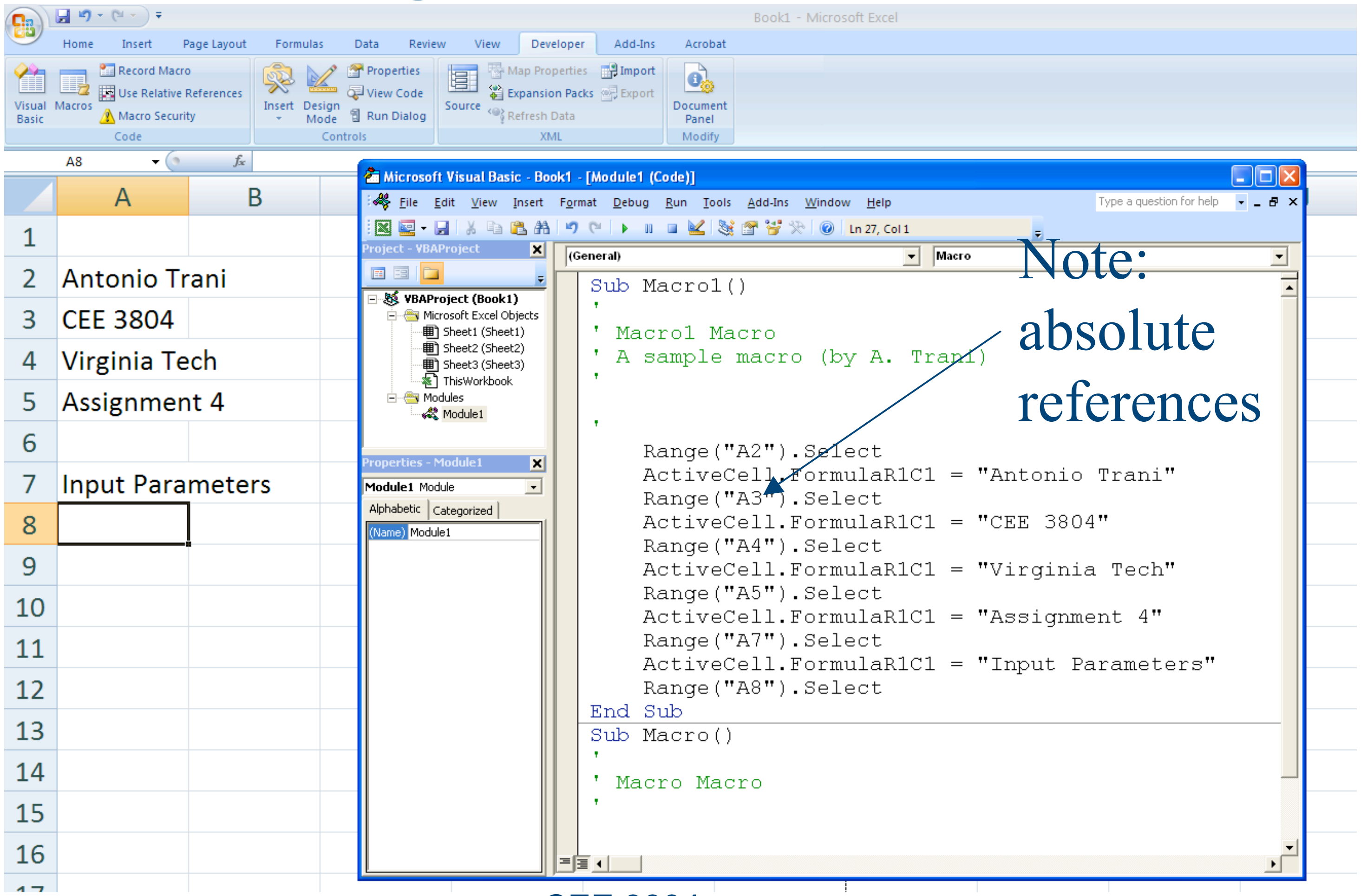


Record Macro
Macro name: Macro1
Shortcut key: Option+Cmd+
Store macro in: This Workbook
Description: Macro recorded 9/18/2007 by toni trani
Cancel OK

Excel 2003
Look under
Tools/Macro/Record New Macro

Macros: A Simple Example

- A macro that creates a template for your homework assignment is shown below

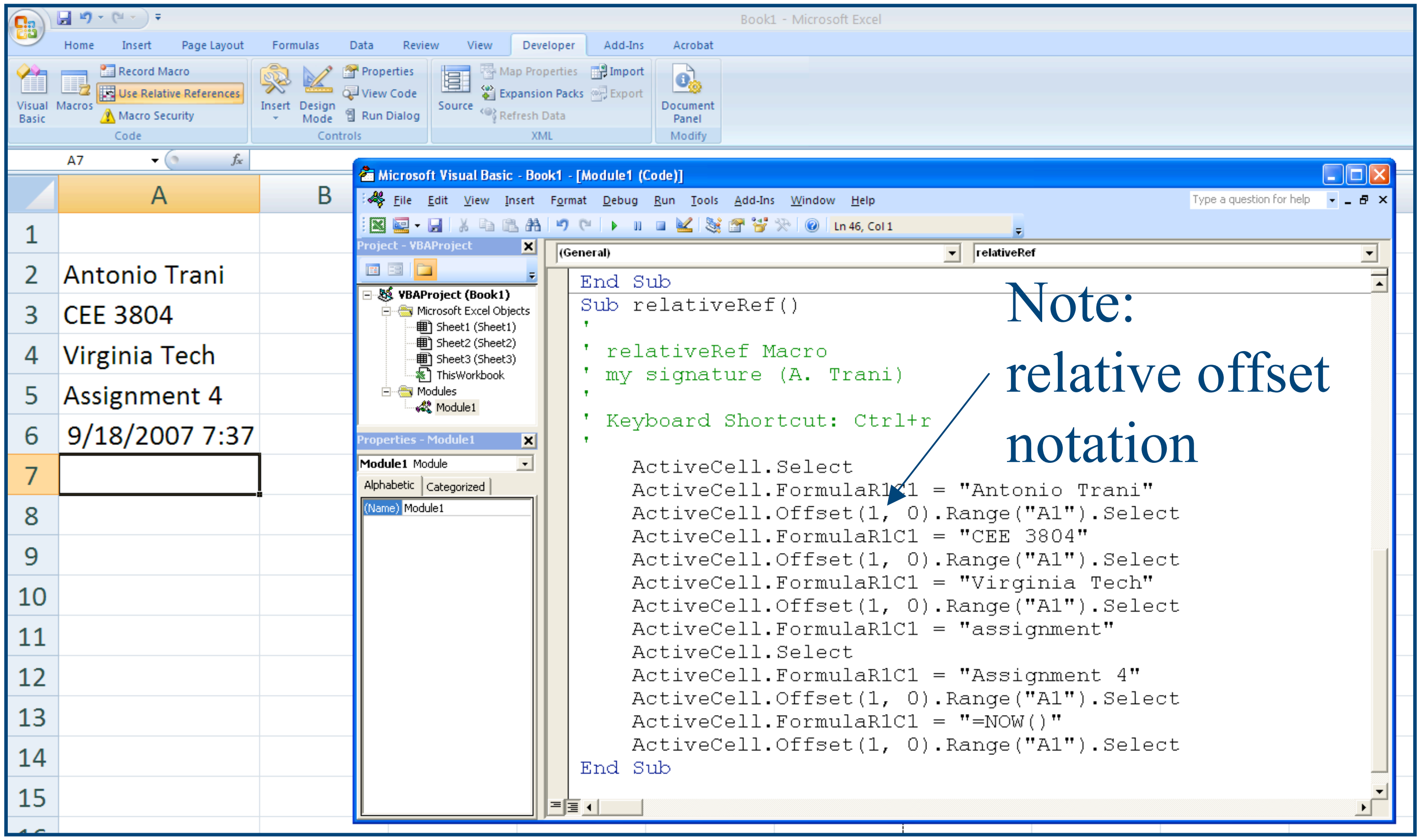


	A	B
1		
2	Antonio Trani	
3	CEE 3804	
4	Virginia Tech	
5	Assignment 4	
6		
7	Input Parameters	
8		

```
Sub Macro1()
'
' Macro1 Macro
' A sample macro (by A. Trani)
'

'
    Range("A2").Select
    ActiveCell.FormulaR1C1 = "Antonio Trani"
    Range("A3").Select
    ActiveCell.FormulaR1C1 = "CEE 3804"
    Range("A4").Select
    ActiveCell.FormulaR1C1 = "Virginia Tech"
    Range("A5").Select
    ActiveCell.FormulaR1C1 = "Assignment 4"
    Range("A7").Select
    ActiveCell.FormulaR1C1 = "Input Parameters"
    Range("A8").Select
End Sub
Sub Macro()
'
' Macro Macro
'
```

Macros: Relative References

- Useful when you need to start the macro at any location in the worksheet



	A
1	
2	Antonio Trani
3	CEE 3804
4	Virginia Tech
5	Assignment 4
6	9/18/2007 7:37
7	

```
End Sub
Sub relativeRef()
'
' relativeRef Macro
' my signature (A. Trani)
'
' Keyboard Shortcut: Ctrl+r
'
    ActiveCell.Select
    ActiveCell.FormulaR1C1 = "Antonio Trani"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "CEE 3804"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Virginia Tech"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "assignment"
    ActiveCell.Select
    ActiveCell.FormulaR1C1 = "Assignment 4"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "=NOW()"
    ActiveCell.Offset(1, 0).Range("A1").Select
End Sub
```

Note: relative offset notation

Macros
Example

- Create a macro called “Title_Logo”:
  - Goes down one row and types the following title:
    - Virginia Tech Civil and Environmental Engineering Department
  - Makes the text bold
  - Inserts the date in the cell below the title using the 04-Mar-00 format
- In Excel 2003 open the Visual Basic editor to view the code:
  - Tools/Macros/Visual Basic Editor or Alt+F11

Macros
Example

```
Sub Title_Logo()
'
' Title_Logo Macro
' Macro recorded 2/7/00
'
' Keyboard Shortcut: Ctrl+t
'
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Virginia Tech Department of
  →       Civil and Environmental Engineering"
    Selection.Font.Bold = True
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "=TODAY()"
    Selection.NumberFormat = "dd-mmm-yy"
End Sub
```

Macros
Storing Macros

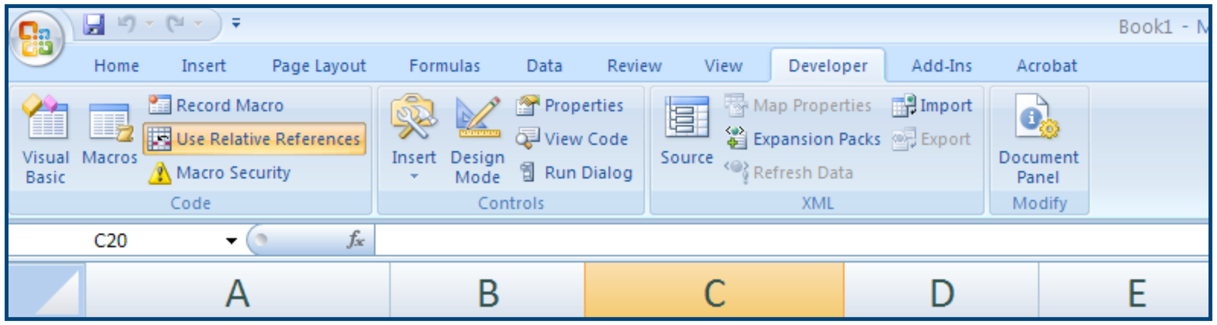
- Macros can be stored:
  - This workbook
    - macros specific to the workbook
  - New workbook
    - Excel generates a new workbook to store the macro
      - advantage: multiple workbook applications can share the same macros
  - Personal macro workbook
    - you are the only person that can use the macros
    - this workbook is a hidden workbook stored in the XLStart folder with the name (personal.xls)
    - macros are available to any open workbooks

Macros
Creating A Custom Command Button

- To create a command button for a macro (Excel 2003):
  - View/Toolbars/Customize
    - In the “Commands” tab click on “Macros”
      - Select “Custom Button” and move the button to the toolbar you want to place it on
      - In “Modify Selection” you can assign a Macro and change the button image
      - In the Name box type the name to be displayed in the button tool tip

Macros
Creating A Custom Command Button

- To create a command button for a macro (Excel 2007):
  - Developer Tab
    - Insert control
      - Select "Button" and move the button to the area in the worksheet you want to place it on
      - Assign the Macro to the button and change the button text information



Macros
Creating A Custom Menu

- To create a Menu Item:
  - View/Toolbars/Customize
    - In the “Commands” tab click on “New Menu”
      - In the new menu select “Macros” and then select “Custom Menu Item”
      - Assign a macro to the menu item and give it a name
      - “&” indicates that an “Alt-key” combination can be used

Editing Macros with the VB Editor
Editor Layout

- The editor consists of three windows:
  - The Project Explorer window
    - whenever a workbook is created a companion VBA project is also created
    - available for each workbook to write code or insert user forms
  - The Properties Window
    - defines the properties of components within a project
    - changes properties at design time
  - The Code Window
    - the Visual Basic Code is stored within a code module
    - the code module is displayed in a code window for editing

VB Basics
Objects, Collections, and Object Models

- **Objects:**
  - elements that represent some part of an application
  - workbook, chart, or form control
- **Collections:**
  - a group of objects usually of the same type
  - group of workbooks
    - Workbooks(1): the first workbook in a sequence of workbooks
- **Object Model:**
  - a hierarchical representation of how the objects and collections are related to each other

VB Basics
Properties, Methods, and Arguments

- Every object has distinct properties & methods
  - A property is an attribute of an object
    - Example: color, font, size, value, etc.
    - ActiveSheet.Name = “Data”
  - A method is an action an object can take
    - Example: printing or copying

      `Application.Quit`

      or

      `ActiveWorkbook.SaveAs “D:\test.xls”`
- Occasionally methods require information:
  - An argument is the information provided to the method
    - Example: `ActiveWorkbook.SaveAs “D:\test.xls”`
    - or `ActiveWorkbook.SaveAs Filename:= “D\test.xls”`

VB Basics
Arguments

- Arguments can be provided in the exact order, or in any order where the argument is preceded by ":="
  - Example:

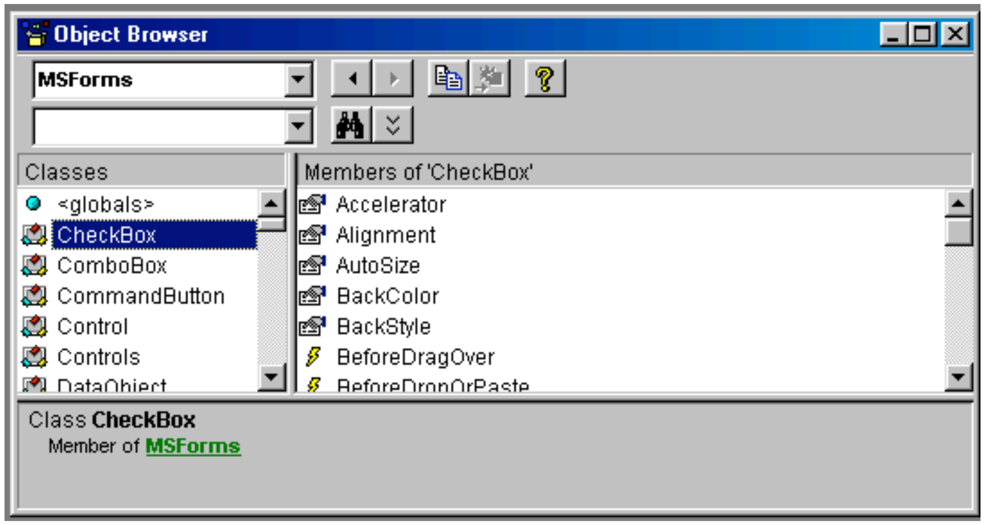
```
ActiveWorkbook.SaveAs FileName:="test.xls"
```

- You can continue a line using the line-continuation character (_)
  - Example:

```
ActiveWorkbook.SaveAs FileName:="test.xls", _
            FileFormat:=xlExcel7
```

VB Basics
Object Libraries

- **The Object Library:**
  - displays object libraries available to the current VBA project
  - press F2 to access the Object Library
  - three main areas:
    - Search area
    - Classes list
    - Members list



Classes	Members of 'CheckBox'
<globals>	Accelerator
CheckBox	Alignment
ComboBox	AutoSize
CommandButton	BackColor
Control	BackStyle
Controls	BeforeDragOver
DataObject	BeforeDropOrPaste

Class **CheckBox**
Member of MSForms

VB Basics
VBA Projects and Components

- VBA creates a project for every open workbook
- contains all of the VBA code written and forms
- forms are custom dialog boxes that allow the user to input information
- code can be written in the code modules behind items
  - items include forms, textboxes, etc.
- code can be written in a standard module
  - ideal for functions that will be shared

VB Basics
Organizing Code

- Within any code module, code is grouped into distinct blocks known as procedures
- A procedure:
    - contains one or more lines of code that accomplish a particular task
    - each line is a statement
    - blank lines are ignored
    - indent lines to make it easier to read the code
    - comments are preceded by colons

VB Basics
Using Code Modules

- To insert a new standard module:
    - In the Visual Basic window
        - Insert/Module
- To change the name:
    - change the properties “Name”
- Group code in a module based on functionality
- To open the code module associated with an application
    - double click on the application

VB Basics
Using Code Windows

- At the top of the window are two drop-down lists:
  - Left box is the Object list
    - lists all objects associated with a window
    - (General) refers to code that does not apply to a specific object
  - Right box is the Procedure list
    - contains a list of all existing procedures within the code module
- Code window is divided into two areas:
  - Declaration and Procedures

VB Basics
Understanding Procedures and Functions

- **Types of procedures:**
  - **Sub procedures:**
    - perform some task
    - begin with a "Sub" statement followed by a unique name
    - and ends with an "End Sub"
    - Can return more than one value via arguments
  - **Function procedures:**
    - perform some task
    - return a single value
    - begin with a "Function" statement followed by a unique name
    - end with an "End Function"
    - set the function name to the value to be returned

VB Basics
Examples of Procedures and Functions

```
Sub ChangeExcelCaption()
   Application.Caption = "My Great Application"
End Sub

Function CalcTakeHome()
    CalcTakeHome = Range("a1") * 0.06
End Function
```

VB Basics
Using Arguments

- The parentheses at the end of the opening statement of a procedure are used to indicate extra information such as arguments:
  - Example:

```
Function CalcTakeHome(Salary)
    CalcTakeHome = Salary * 0.06
End Function
```

  -
  - To access a custom function:
    - Insert/Function/User Defined

VB Basics
Calling Procedures

- To call a sub procedure:
  - type name of sub procedure followed by a space and the name of the argument
    - Example:

```
CalcTakeHome RealSalary
```

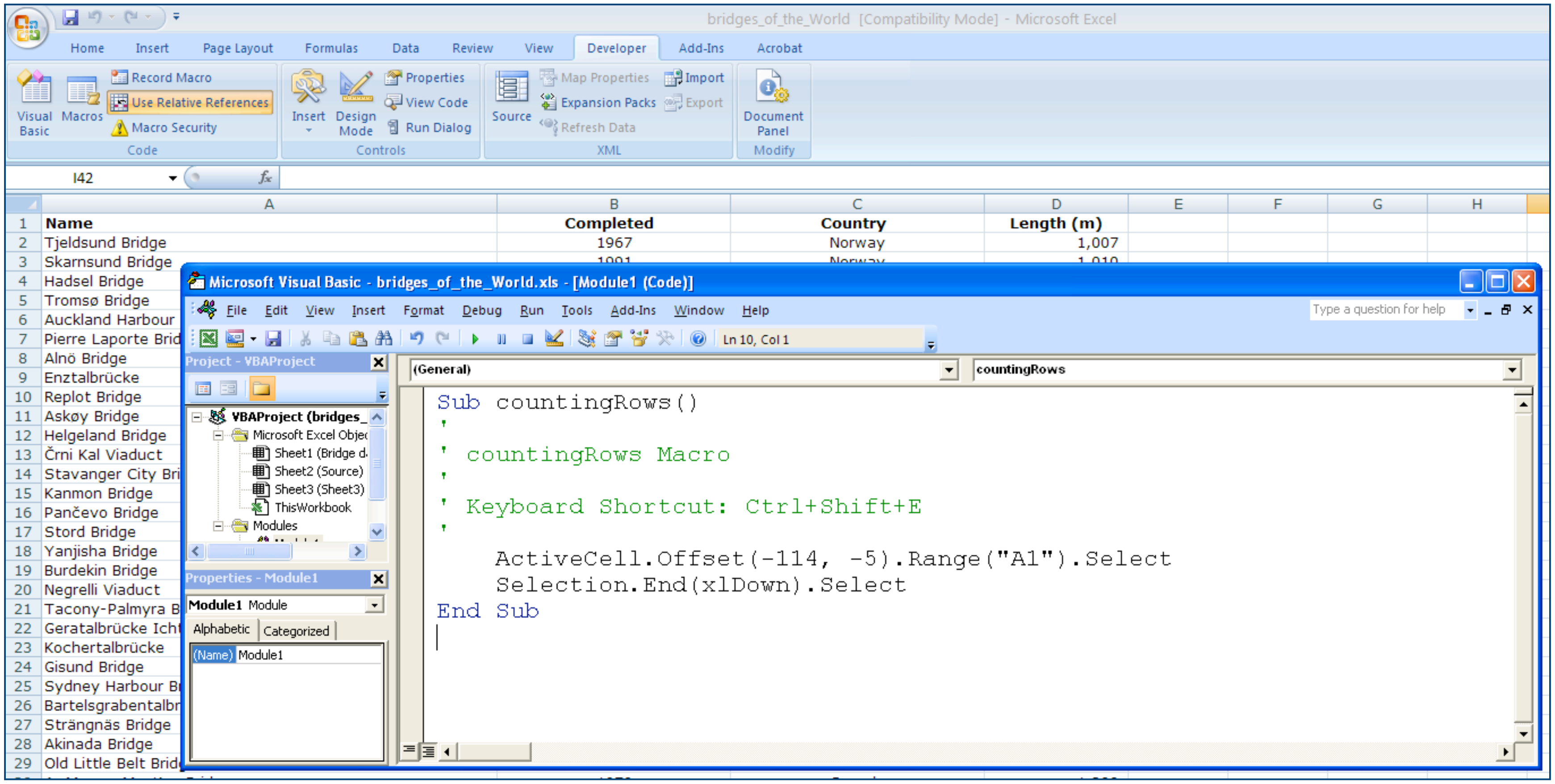
- To call a function:
  - need to provide a variable to store the value
    - Example:

```
RealSalary = CalcTakeHome(50000)
```

VBA Example : Counter of Data Macro

- See example in Section 3.3 on page 27 (Chapra's textbook)
- Example creates a macro to calculate the number of rows in a data set
- Uses a macro to get VBA code to move from an initial position in the worksheet to an ending position (sub countingRows)
- Use the "bridges_of_the_world.xls" file
- A second sub called countCells computes the number of cells and displays the result in a message box

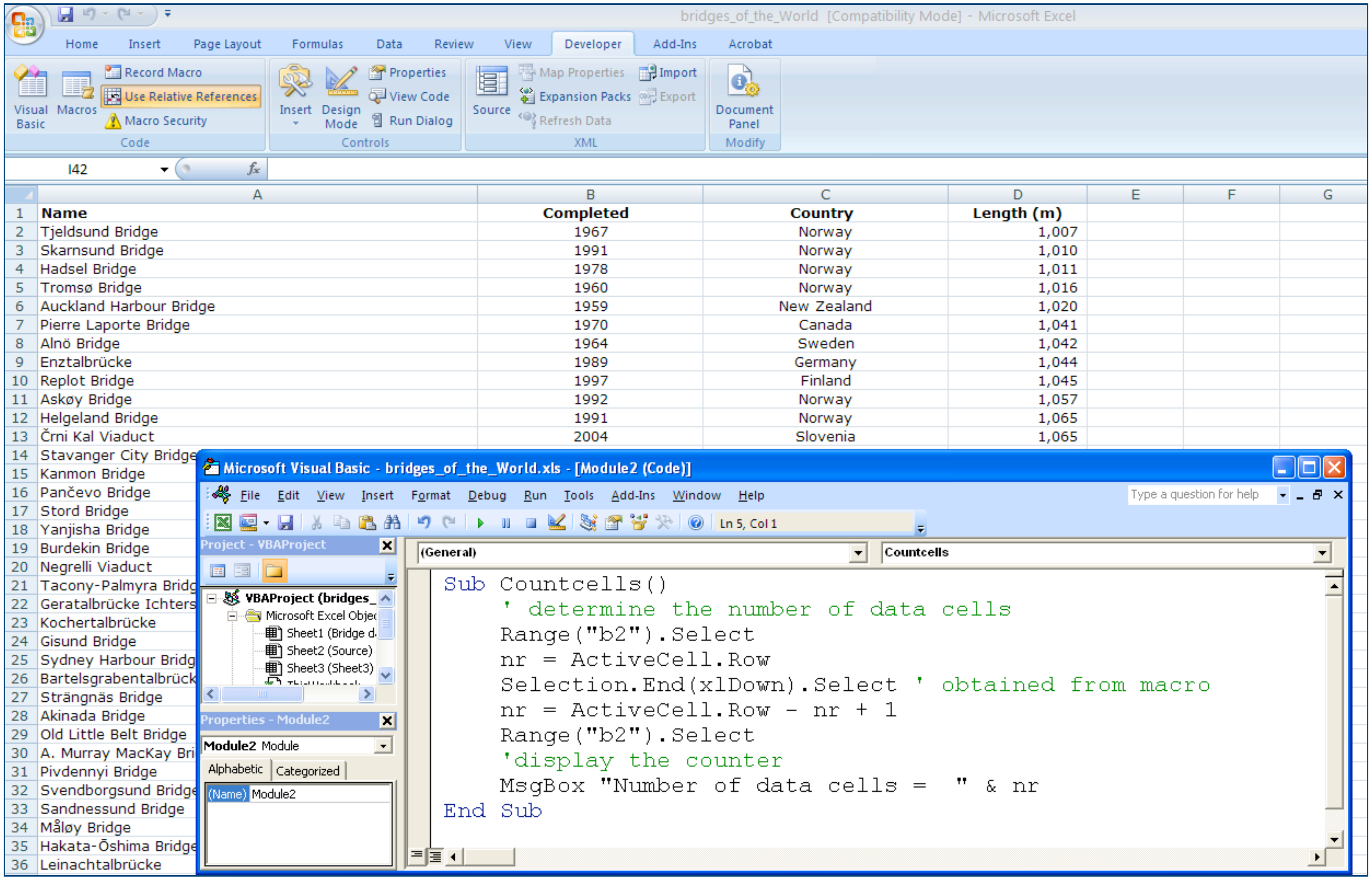
VBA Macro Example (countingRows)



	A	B	C	D
1	Name	Completed	Country	Length (m)
2	Tjeldsund Bridge	1967	Norway	1,007
3	Skarnsund Bridge			
4	Hadsel Bridge			
5	Tromsø Bridge			
6	Auckland Harbour			
7	Pierre Laporte Brid			
8	Alnö Bridge			
9	Enztalbrücke			
10	Replot Bridge			
11	Askøy Bridge			
12	Helgeland Bridge			
13	Črni Kal Viaduct			
14	Stavanger City Bri			
15	Kanmon Bridge			
16	Pančevo Bridge			
17	Stord Bridge			
18	Yanjisha Bridge			
19	Burdekin Bridge			
20	Negrelli Viaduct			
21	Tacony-Palmyra B			
22	Geratalbrücke Icht			
23	Kochertalbrücke			
24	Gisund Bridge			
25	Sydney Harbour Br			
26	Bartelsgrabentalbr			
27	Strängnäs Bridge			
28	Akinada Bridge			
29	Old Little Belt Brid			

```
Sub countingRows()
'
' countingRows Macro
'
' Keyboard Shortcut: Ctrl+Shift+E
'
    ActiveCell.Offset(-114, -5).Range("A1").Select
    Selection.End(xlDown).Select
End Sub
```

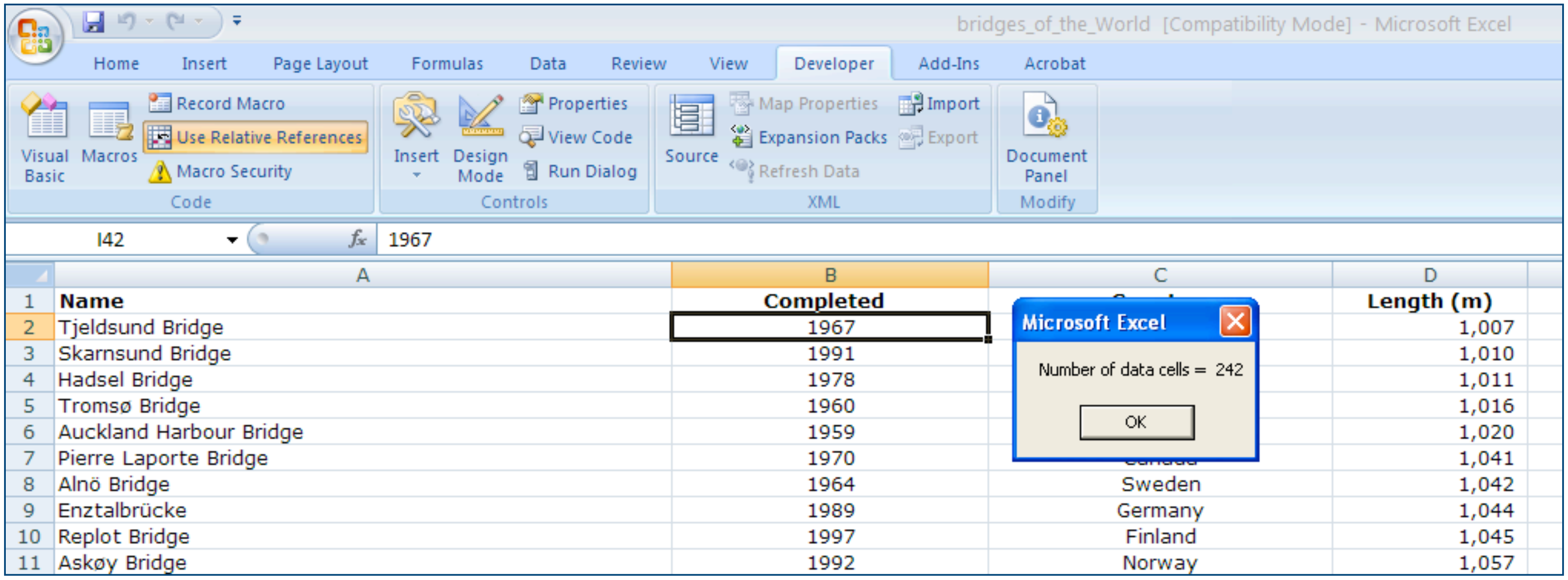
VBA Macro Example (Countcells)



	A	B	C	D
1	**Name**	**Completed**	**Country**	**Length (m)**
2	Tjeldsund Bridge	1967	Norway	1,007
3	Skarnsund Bridge	1991	Norway	1,010
4	Hadsel Bridge	1978	Norway	1,011
5	Tromsø Bridge	1960	Norway	1,016
6	Auckland Harbour Bridge	1959	New Zealand	1,020
7	Pierre Laporte Bridge	1970	Canada	1,041
8	Alnö Bridge	1964	Sweden	1,042
9	Enztalbrücke	1989	Germany	1,044
10	Replot Bridge	1997	Finland	1,045
11	Askøy Bridge	1992	Norway	1,057
12	Helgeland Bridge	1991	Norway	1,065
13	Črni Kal Viaduct	2004	Slovenia	1,065
14	Stavanger City Bridge			
15	Kanmon Bridge			
16	Pančevo Bridge			
17	Stord Bridge			
18	Yanjisha Bridge			
19	Burdekin Bridge			
20	Negrelli Viaduct			
21	Tacony-Palmyra Bridg			
22	Geratalbrücke Ichters			
23	Kochertalbrücke			
24	Gisund Bridge			
25	Sydney Harbour Bridg			
26	Bartelsgrabentalbrück			
27	Strängnäs Bridge			
28	Akinada Bridge			
29	Old Little Belt Bridge			
30	A. Murray MacKay Bri			
31	Pivdennyi Bridge			
32	Svendborgsund Bridge			
33	Sandnessund Bridge			
34	Måløy Bridge			
35	Hakata-Ōshima Bridge			
36	Leinachtalbrücke			

```
Sub Countcells()
    ' determine the number of data cells
    Range("b2").Select
    nr = ActiveCell.Row
    Selection.End(xlDown).Select ' obtained from macro
    nr = ActiveCell.Row - nr + 1
    Range("b2").Select
    'display the counter
    MsgBox "Number of data cells =  " & nr
End Sub
```

VBA Macro Example : Running



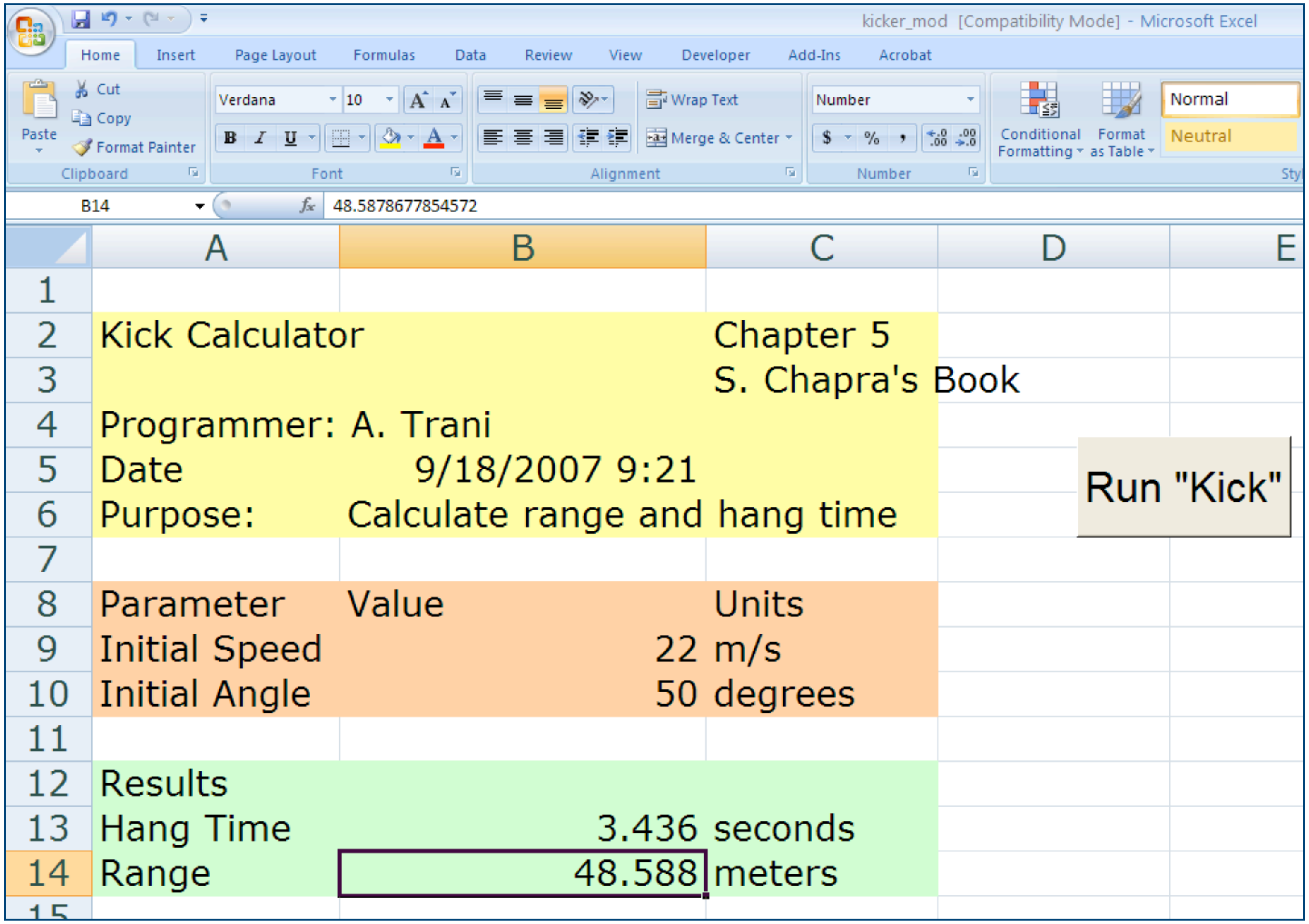
Number of data cells = 242

	A	B	C	D
1	Name	Completed		Length (m)
2	Tjeldsund Bridge	1967		1,007
3	Skarnsund Bridge	1991		1,010
4	Hadsel Bridge	1978		1,011
5	Tromsø Bridge	1960		1,016
6	Auckland Harbour Bridge	1959		1,020
7	Pierre Laporte Bridge	1970		1,041
8	Alnö Bridge	1964	Sweden	1,042
9	Enztalbrücke	1989	Germany	1,044
10	Replot Bridge	1997	Finland	1,045
11	Askøy Bridge	1992	Norway	1,057

VBA Example : Kicker

- Section 5.1 in Chapra's textbook (see pages 40-47)
- Projectile motion example
- Illustrates how a sub calls another sub
- Illustrates how a sub generates multiple results and passes them to another one
- Sub kickCalculation (main routine)
- Sub calculationForKicker (called from kickCalculation)

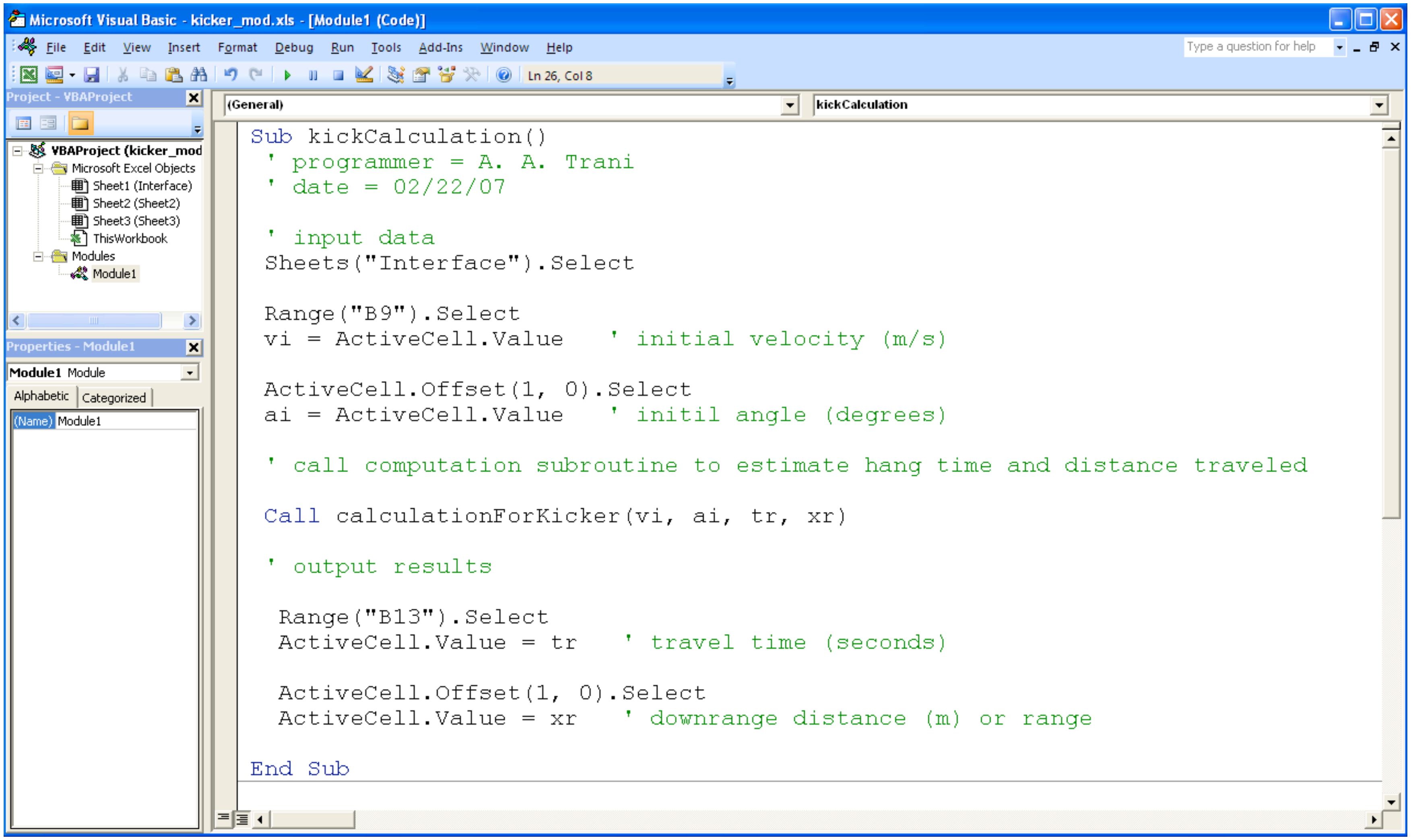
VBA Example : Kicker Worksheet


kicker_mod [Compatibility Mode] - Microsoft Excel
Home Insert Page Layout Formulas Data Review View Developer Add-Ins Acrobat
Paste Cut Copy Format Painter Clipboard
Verdana 10 Font
Wrap Text Merge & Center Alignment
Number Number
Conditional Formatting Format as Table
Normal Neutral
Run "Kick"

B14 fx 48.5878677854572

	A	B	C	D	E
1					
2	Kick Calculator		Chapter 5		
3			S. Chapra's Book		
4	Programmer: A. Trani				
5	Date	9/18/2007 9:21			
6	Purpose:	Calculate range and hang time			
7					
8	Parameter	Value	Units		
9	Initial Speed	22	m/s		
10	Initial Angle	50	degrees		
11					
12	Results				
13	Hang Time	3.436	seconds		
14	Range	48.588	meters		

Sub : kickCalculation



```
Sub kickCalculation()
 ' programmer = A. A. Trani
 ' date = 02/22/07

 ' input data
 Sheets("Interface").Select

 Range("B9").Select
 vi = ActiveCell.Value    ' initial velocity (m/s)

 ActiveCell.Offset(1, 0).Select
 ai = ActiveCell.Value    ' initil angle (degrees)

 ' call computation subroutine to estimate hang time and distance traveled

 Call calculationForKicker(vi, ai, tr, xr)

 ' output results

  Range("B13").Select
  ActiveCell.Value = tr    ' travel time (seconds)

  ActiveCell.Offset(1, 0).Select
  ActiveCell.Value = xr    ' downrange distance (m) or range

End Sub
```

Observations about kickCalculation

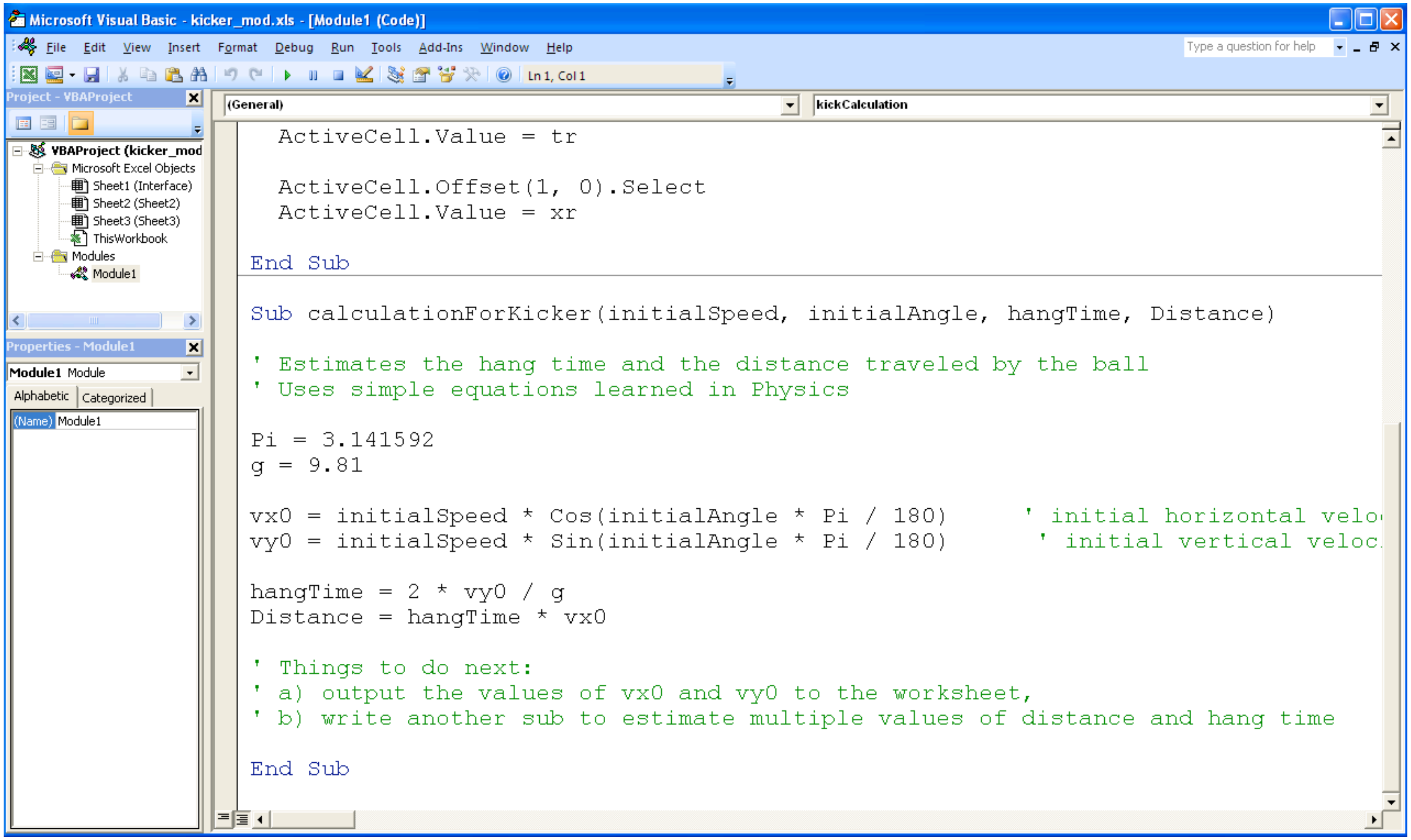
- The subroutine reads two values vi and ai in cells B9 and B10

```
' input data
Sheets("Interface").Select
Range("B9").Select
vi = ActiveCell.Value
```

…..

- Then a call to subroutine calculationForKicker is made
- This sub call provides two input values (vi and ai)
- In return the sub provides two output values (tr and xr)
- The values of tr (hang time) and xr (distance) are then inserted back to the worksheet in cells B13 and B14

Sub : calculationForKicker



```
  ActiveCell.Value = tr

  ActiveCell.Offset(1, 0).Select
  ActiveCell.Value = xr

End Sub

Sub calculationForKicker(initialSpeed, initialAngle, hangTime, Distance)

' Estimates the hang time and the distance traveled by the ball
' Uses simple equations learned in Physics

Pi = 3.141592
g = 9.81

vx0 = initialSpeed * Cos(initialAngle * Pi / 180)      ' initial horizontal velo
vy0 = initialSpeed * Sin(initialAngle * Pi / 180)       ' initial vertical veloc

hangTime = 2 * vy0 / g
Distance = hangTime * vx0

' Things to do next:
' a) output the values of vx0 and vy0 to the worksheet,
' b) write another sub to estimate multiple values of distance and hang time

End Sub
```

Things to Observe

- The definition of the sub is:

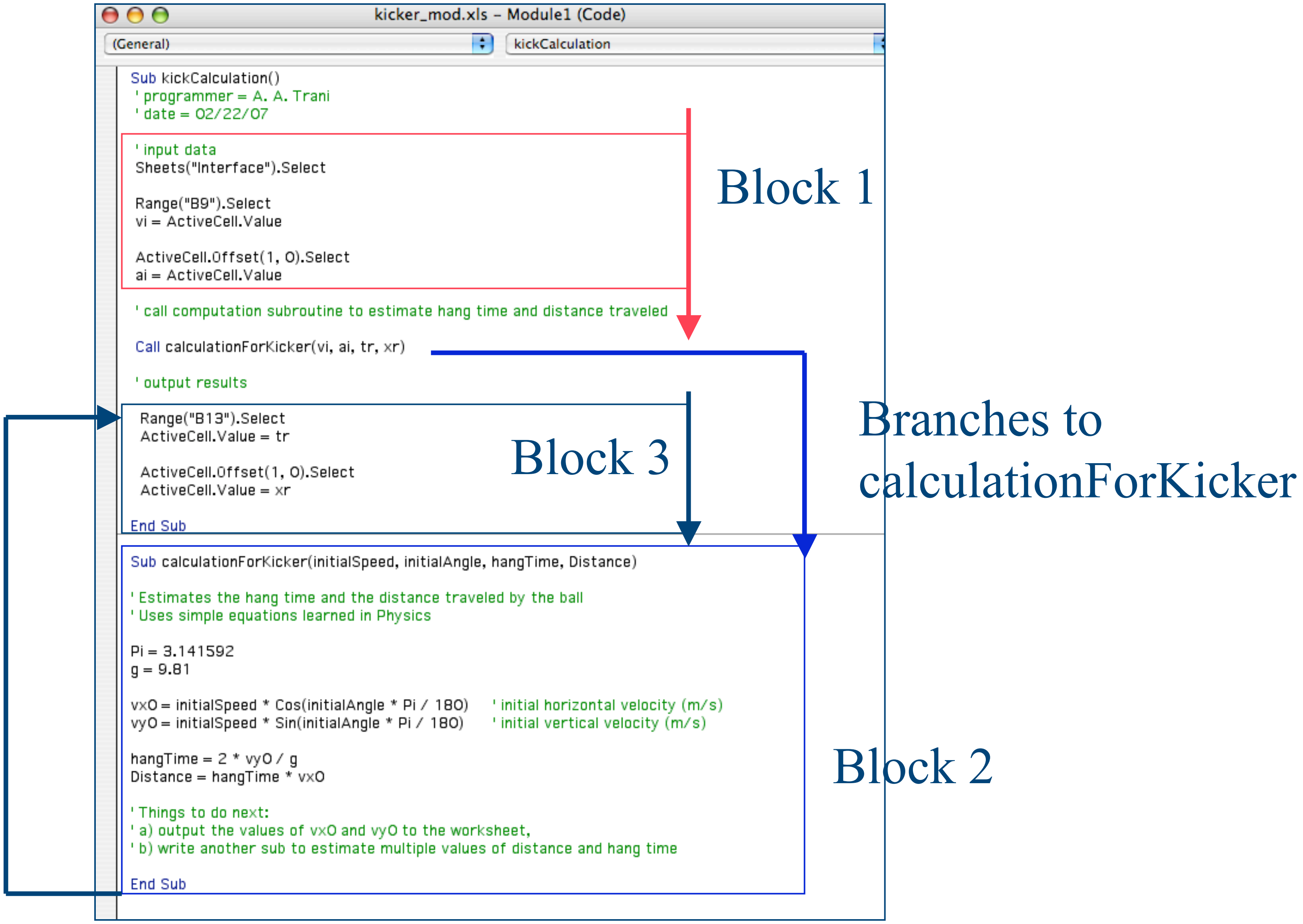
  Sub calculationForKicker(initialSpeed, initialAngle, hangTime, Distance)

- Yet the sub is called using the following statement

  Call calculationForKicker(vi, ai, tr, xr)

- In this example, the main sub kickCalculation contains the variable names that will be inserted in the worksheet
- The number of arguments in the sub calculationForKicker and kickCalculation are the same
- The variable names initialSpeed, initialAngle, hangTime and Distance are placeholders that get to be replaced by variable names contained in the sub that calls calculationForKicker

Order of Execution



kicker_mod.xls – Module1 (Code)

(General) kickCalculation

```
Sub kickCalculation()
 ' programmer = A. A. Trani
 ' date = 02/22/07

 ' input data
 Sheets("Interface").Select

 Range("B9").Select
 vi = ActiveCell.Value

 ActiveCell.Offset(1, 0).Select
 ai = ActiveCell.Value

 ' call computation subroutine to estimate hang time and distance traveled

 Call calculationForKicker(vi, ai, tr, xr)

 ' output results
  Range("B13").Select
  ActiveCell.Value = tr

  ActiveCell.Offset(1, 0).Select
  ActiveCell.Value = xr

End Sub

Sub calculationForKicker(initialSpeed, initialAngle, hangTime, Distance)

' Estimates the hang time and the distance traveled by the ball
' Uses simple equations learned in Physics

Pi = 3.141592
g = 9.81

vx0 = initialSpeed * Cos(initialAngle * Pi / 180)    ' initial horizontal velocity (m/s)
vy0 = initialSpeed * Sin(initialAngle * Pi / 180)    ' initial vertical velocity (m/s)

hangTime = 2 * vy0 / g
Distance = hangTime * vx0

' Things to do next:
' a) output the values of vx0 and vy0 to the worksheet,
' b) write another sub to estimate multiple values of distance and hang time

End Sub
```

VB Basics
Event Procedures

- **Definition:**
  - **event procedures are procedures that are used with events**
- **Event procedures are stored in the code module associated with the object:**
  - **to add code to the Open event of the active workbook, you will use the code module behind ThisWorkbook**
- **Event procedure name is a combination of:**
  - **object name, “_”, and event name**
    - **Example:**

```
Private Sub Workbook_BeforePrint()
```

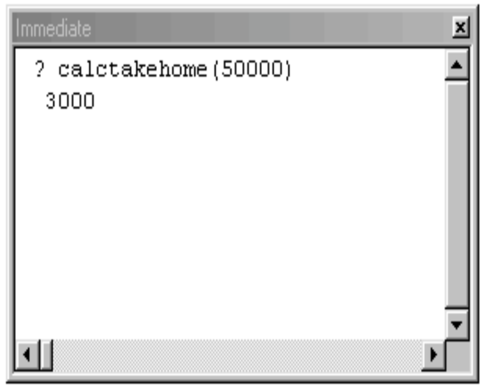
    - **In the example `Workbook` is the object name and `BeforePrint()` is the event name. This event procedure is called before the Workbook is printed**

VB Basics
Running and Testing Procedures

- Can run a procedure within the VB window:
  - Run/Sub or F5
- Two methods for testing procedures:
  - Run your procedure
  - Use the immediate window (View/Immediate Window)
    - Example:

```
? CalcTakeHome(50000)
```

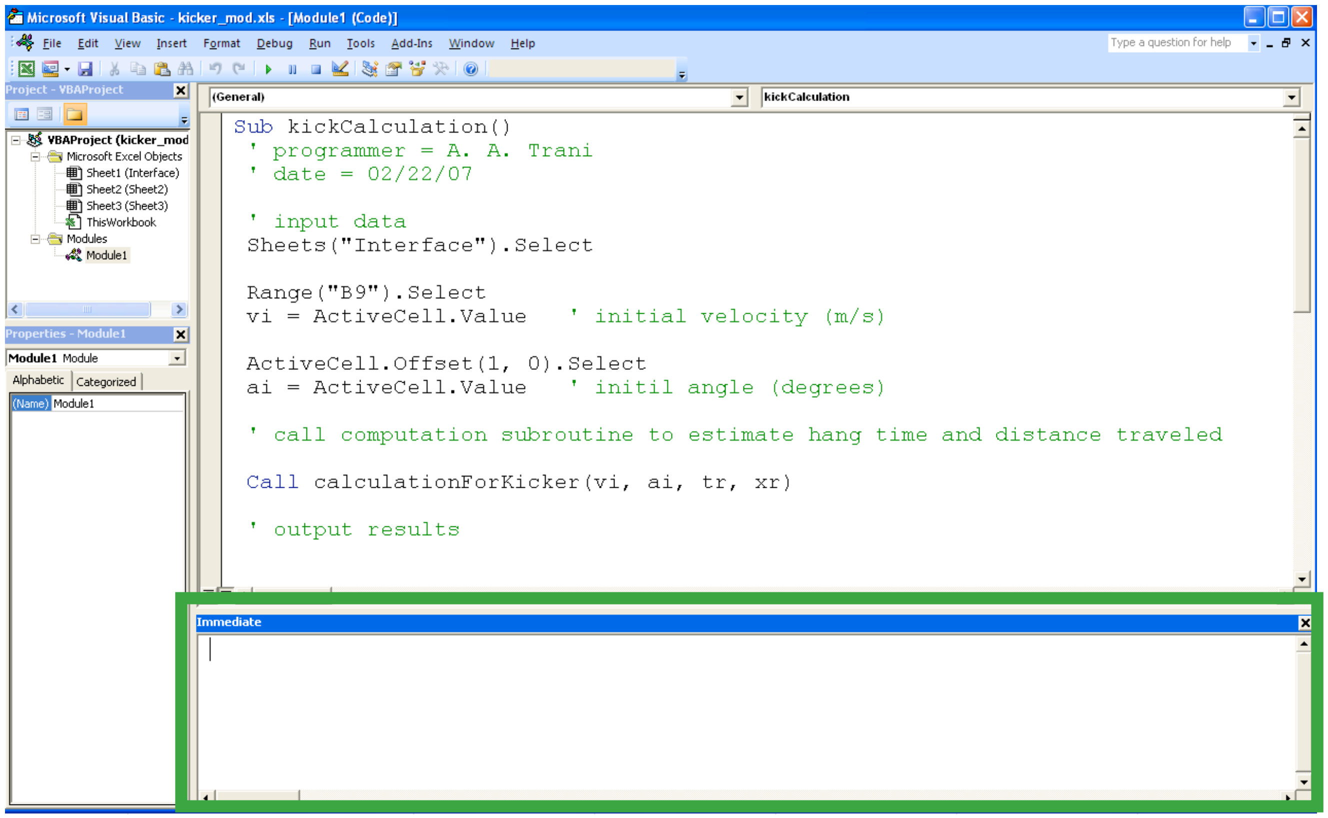
```
Function CalcTakeHome(Salary)
    CalcTakeHome = Salary * 0.06
End Function
```



```
? calctakehome(50000)
 3000
```

VBA Testing: Immediate Window (Excel 2007)

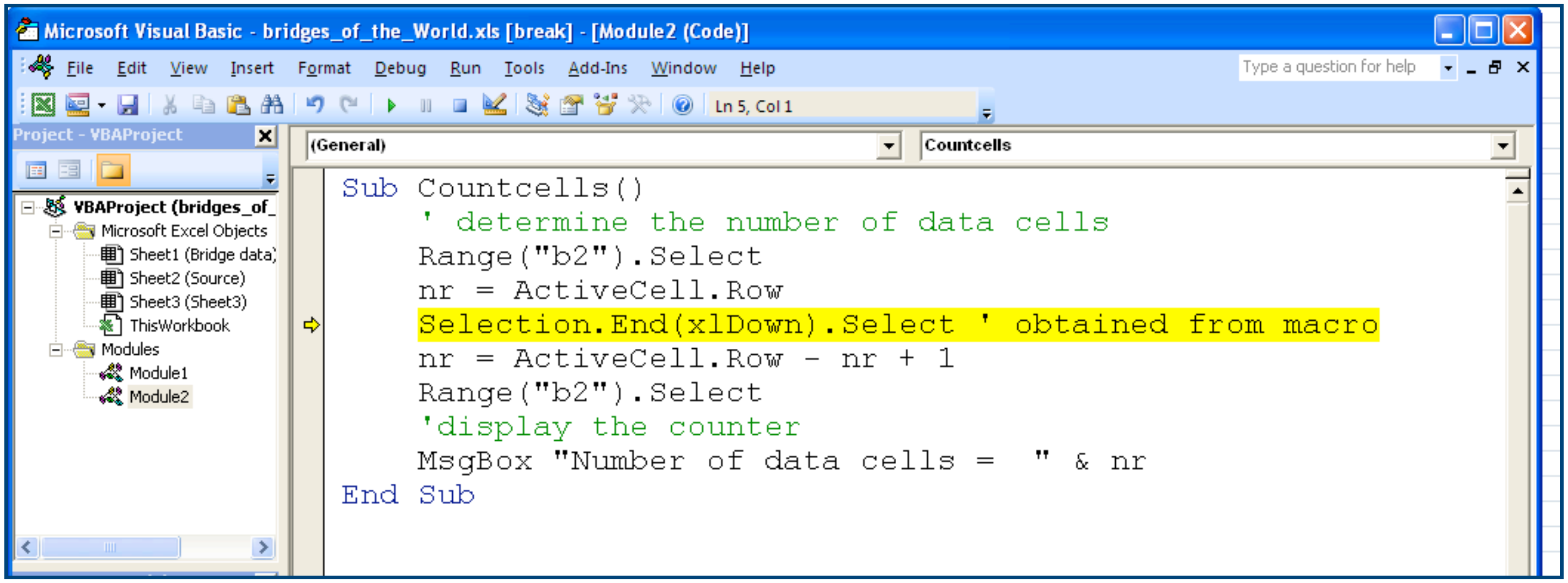
- In VBE editor
- Control + G to active the immediate window in Excel 2007



```
Sub kickCalculation()
 ' programmer = A. A. Trani
 ' date = 02/22/07

 ' input data
Sheets("Interface").Select

Range("B9").Select
vi = ActiveCell.Value    ' initial velocity (m/s)

ActiveCell.Offset(1, 0).Select
ai = ActiveCell.Value    ' initil angle (degrees)

 ' call computation subroutine to estimate hang time and distance traveled

Call calculationForKicker(vi, ai, tr, xr)

 ' output results
```

VBA Basics: debugger in VBA

- VBA has a fully functional debugger to help out streamline your programs
- Can "step-in" the code line by line to see your intermediate calculations



```
Sub Countcells()
    ' determine the number of data cells
    Range("b2").Select
    nr = ActiveCell.Row
    Selection.End(xlDown).Select ' obtained from macro
    nr = ActiveCell.Row - nr + 1
    Range("b2").Select
    'display the counter
    MsgBox "Number of data cells =  " & nr
End Sub
```

VB Basics
Variables and Constants

- Definition:
  - Variables are named locations in memory
- Need to declare variables explicitly:
  - defines type of data, procedures that use data, and avoids errors
  - a variable declared within a procedure is a local variable

```
Dim [variable name] As [data type]
```

  - use “public” or “private” to share variables
  - only public variables can be used for other code modules

VB Basics

Data Types

- Type of Data:
  - Byte: 0 to 255
  - Integer: -32,768 to 32,767
  - Long: -2,000m to 2000m
  - Single: -3.4E38 to 3.4E38
  - Double: -1.8E308 to 1.8E308
  - Boolean: -1 or 0
  - String: 0 to 2 billion characters
  - Variant: Anything (including special values and Null)

VB Basics
Variable Declaring Variables and Objects

- To force variable declaration:
    - Option Explicit at top of module
    - In the Options box enable “Require Variable Declaration”
- Object variables:
    - Special types of variables directed at objects rather than data
        - nickname for object

```
Dim app as Application
```

        - initializing variable

```
Set app = Application
```

VB Basics
Constants

- **Definition:**
  - **similar to variables but can only be filled with data once**
- **Built-in constants:**
  - **`vbRed`: refers to the color red**
- **Constant declaration:**
  - **`Const` [*name of constant*] = [*value*] `As` [*data type*]**
  - **Create constants in capital letters to distinguish from other variables**

VB Basic User Interaction
Displaying a Message

- The Msgbox function can be used to display information:

```
MsgBox "Download Complete."
```

- MsgBox function arguments:
  - `MsgBox`(*Prompt*, *Buttons*, *Title, HelpFile, Context*)
    - Prompt: Message displayed to user
    - Buttons: a combination of numerical constants
      - buttons, icon, default button, modality, and other
    - Title: indicates the string value that appears in the title bar
    - HelpFile and Context: provide help information

VB Basic User Interaction
Displaying a Message: Button Argument

- The Buttons option includes:
  - Buttons:
    - `vbOkOnly, vbOkCancel, vbAbortRetryIgnore, vbYesNoCancel, vbYesNo, vbRetryCancel`
  - Icon:
    - `vbCritical, vbQuestion, vbExclamation, vbInformation`
  - Default Button:
    - `vbDefaultButton1, vbDefaultButton2, vbDefaultButton3, vbDefaultButton4`
  - Modality:
    - `vbApplicationModal`: user may respond before using any application
  - Other:
    - `vbMsgBoxHelpButton, vbMsgBoxSetForeground, vbMsgBoxRight`

VB Basic User Interaction
Returning Button Constants

- To know which button was clicked:
  - MsgBox returns a constant value that indicates which button was clicked
    - vbOK: OK button clicked
    - vbCancel: Cancel button clicked
    - vbAbort: Abort button clicked
    - vbRetry: Retry button clicked
    - vbIgnore: Ignore button clicked
    - vbYes: Yes button clicked or MsgBox = 6
    - vbNo: No button clicked or MsgBox = 7

VB Basic User Interaction
Message Box Example

- **Example:**

```
Sub ChangeExcelCaption()
   Dim intResponse As Integer
'-------------------------------------------------------
  intResponse = MsgBox("Would you like to change the worksheet title?", _
                 vbQuestion + vbYesNo + vbApplicationModal + vbMsgBoxHelpButton,
_
                 "Change Excel Caption")

  If (intResponse = vbYes) Then
    Application.Caption = "My Great Application"
  End If
End Sub
```

VB Basic User Interaction
Getting Data from Users

- The InputBox function retrieves information from user:
  - InputBox(*prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context]*)
    - prompt: is the message that is displayed in the dialog box
    - title: the string value in the title bar of the message box
    - default: displays default text
    - xpos: position of left edge of box from left edge of screen in twips (default is centered horizontally)
    - ypos: similar to xpos except for vertical position
    - helpfile and context: provide help information

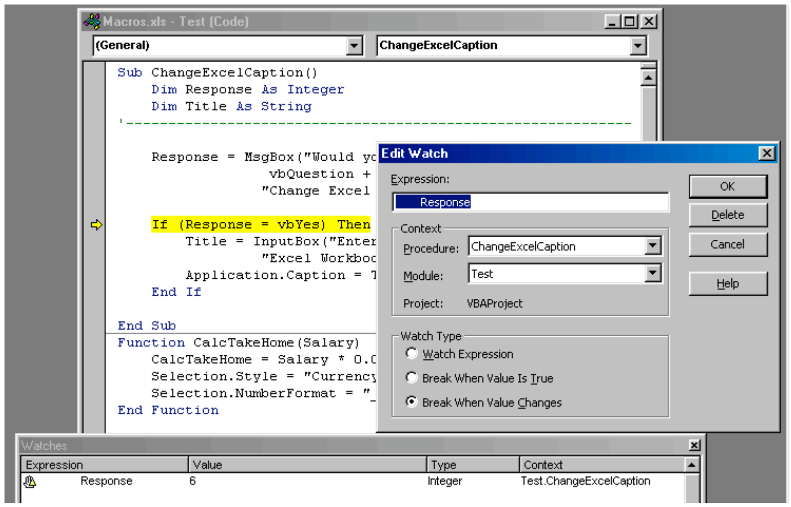
VB Basic User Interaction
InputBox Function Example

- **Example:**

```
Sub ChangeExcelCaption()
   Dim Response As Integer
   Dim Title As String
   '-------------------------------------------------------------------
  Response = MsgBox("Would you like to change the worksheet title?", _
             vbQuestion + vbYesNo + vbApplicationModal + _
             vbMsgBoxHelpButton, "Change Excel Caption")
  If (Response = vbYes) Then
        Title = InputBox("Enter Caption?", "Change Excel Caption", _
                 "Excel Workbook")
        Application.Caption = Title
    End If
End Sub
```

VB Debugging
Breakpoints and Watch Windows

- Insert a breakpoint to stop program at specific location:
  - view variable values by placing mouse on variable
  - use immediate window to print out values of variables
  - create watch windows:
    - automatically insert a break when value changes



```
Sub ChangeExcelCaption()
    Dim Response As Integer
    Dim Title As String
'------------------------------------------------------------

    Response = MsgBox("Would yc
                vbQuestion +
               "Change Excel

    If (Response = vbYes) Then
        Title = InputBox("Enter
               "Excel Workboc
        Application.Caption = T
    End If

End Sub
Function CalcTakeHome(Salary)
    CalcTakeHome = Salary * 0.C
    Selection.Style = "Currency
    Selection.NumberFormat = "_
End Function
```

Expression: Response

Context
Procedure: ChangeExcelCaption
Module: Test
Project: VBAProject

Watch Type
( ) Watch Expression
( ) Break When Value Is True
[✓ Break When Value Changes]

Expression	Value	Type	Context
Response	6	Integer	Test.ChangeExcelCaption

VB Basic Coding
Branching in Code: Overview

- **Different branching are available:**
  - **If, End If**
    - **Single or multiple conditions**
      - **If [*statement is true*] Then**
        ElseIf [*alternative statement is true*] Then
        Else
      - **End If**
  - **Select Case, End Select:**
    - **Single condition with multiple results**
      - **Select Case [*some expression*]**
        Case [*result 1*]
        Case Else
      - **End Select**

VB Basic Coding
Branching Example

```
Sub CalcWeekDay()
    Dim strDate As String

    strDate = InputBox("Enter a date using mm/dd/yy format:", _
                "Date Input", Date)
    Select Case WeekDay(strDate)
    Case vbMonday To vbThursday
        MsgBox (strDate & " falls on Monday thru Thursday ...")
    Case vbFriday
        MsgBox (strDate & " is a Friday!")
    Case Else
        MsgBox (strDate & " is a weekend day.")
    End Select
End Sub
```

VB Basic Coding
Repetition: Do … Loop

- **Different ways of implementing repetition:**
  - Do… Loop
    - While [condition is TRUE]: loop continues as long as condition is true
    - Until [condition is TRUE]: loop continues as long as the expression evaluates to false
    - Two ways of coding:

```
Do While [condition] or Until [condition]
        code to be repeated
Loop


Do
        code to be repeated
Loop While [condition] or Until [condition]
```

VB Basic Coding
Repetition: For… Next

- Another way of repeating code:
  - Standard:

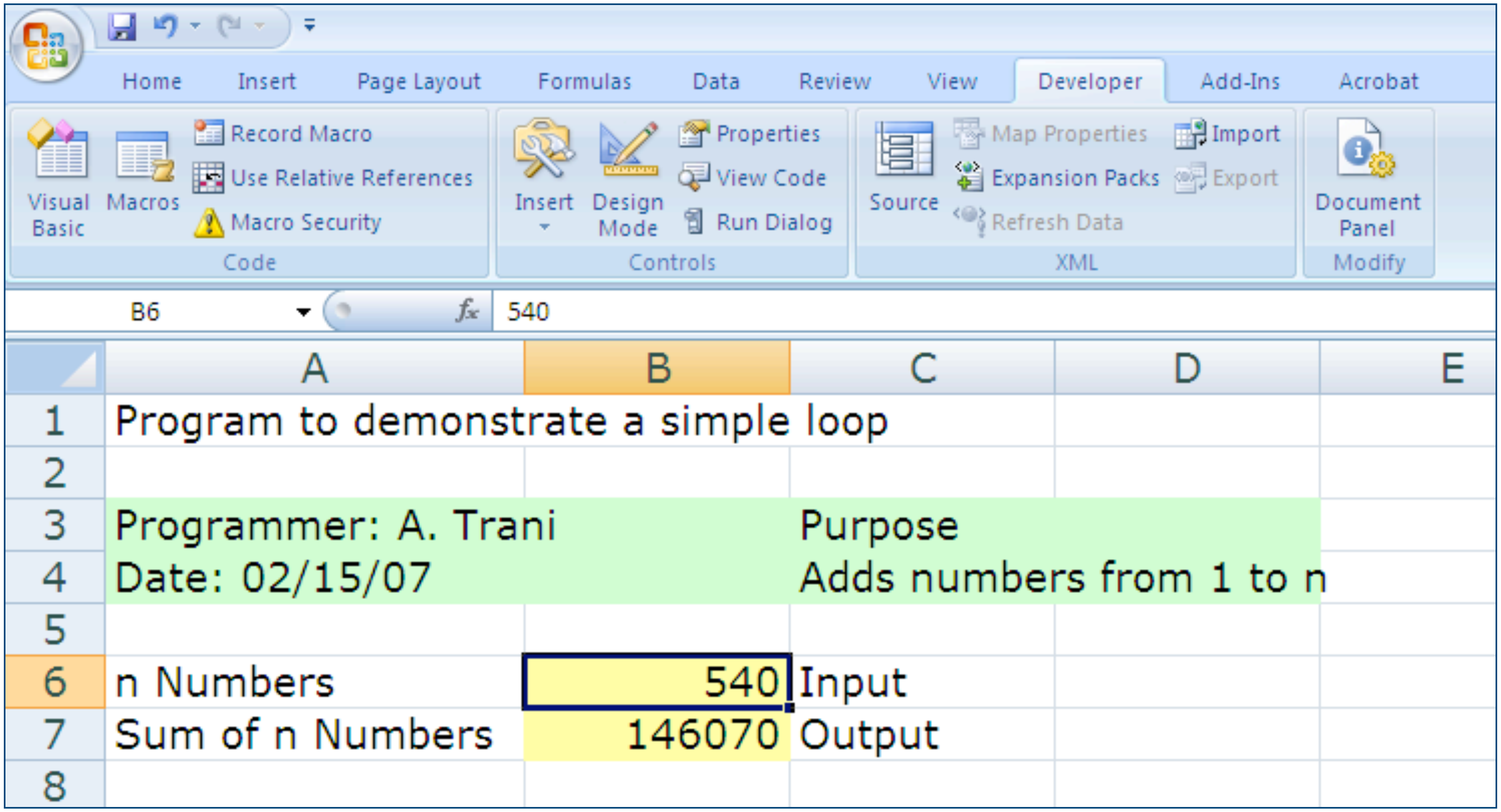
    ```
    For [counter variable] = [start value] To [end value]
        code to be repeated
    Next [counter variable]
    ```

  - Optional:

    ```
    For [counter variable] = [start value] To [end value] Step [increment]
        code to be repeated
    Next [counter variable]
    ```

  - For Each… Next:
    - allows you to loop through the collection of objects without knowing the precise number of objects

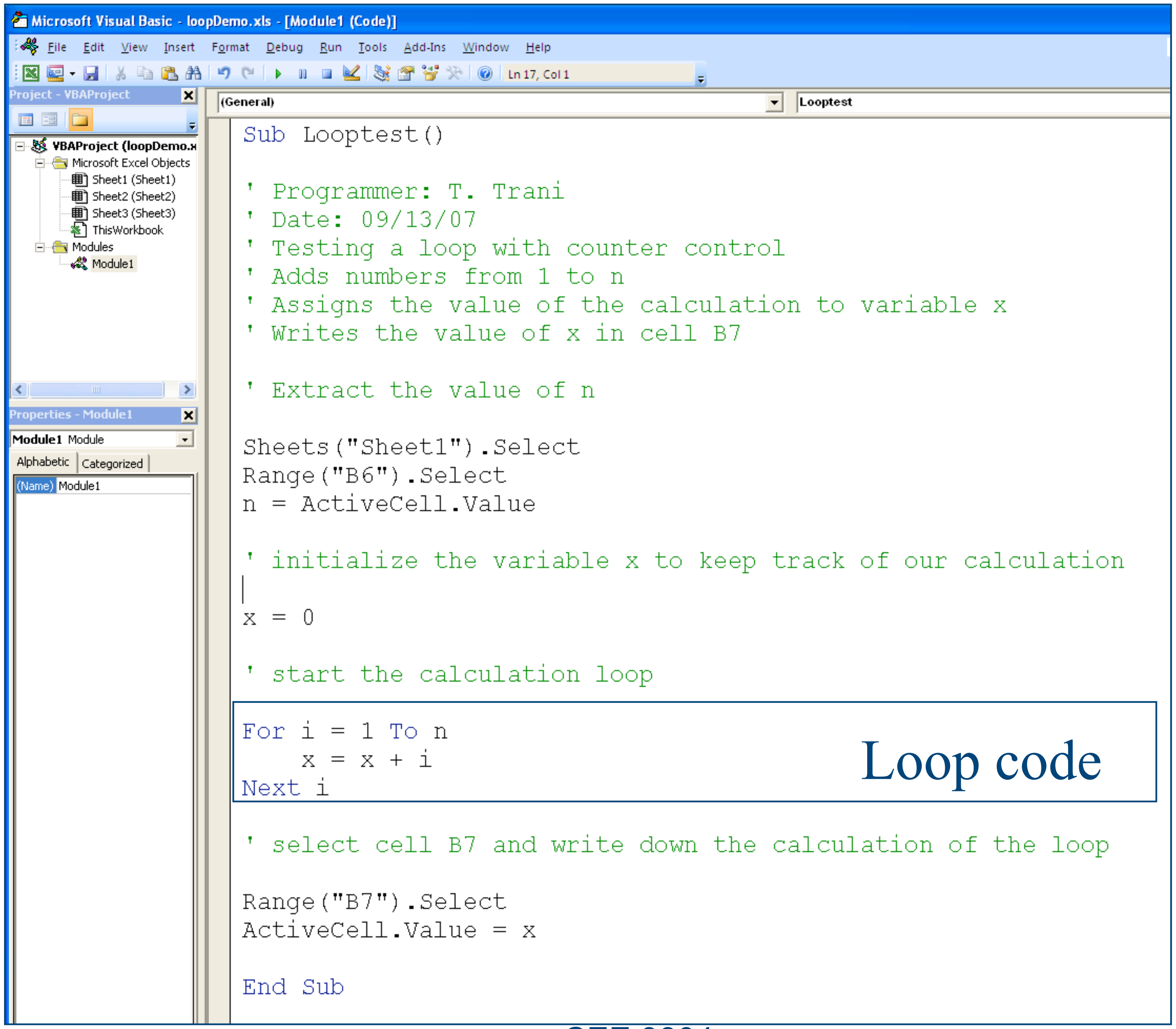
First Program with a Loop



B6 | 540

	A	B	C	D	E
1	Program to demonstrate a simple loop				
2					
3	Programmer: A. Trani		Purpose		
4	Date: 02/15/07		Adds numbers from 1 to n		
5					
6	n Numbers	540	Input		
7	Sum of n Numbers	146070	Output		
8					

The VBA Code Behind



```
Sub Looptest()

' Programmer: T. Trani
' Date: 09/13/07
' Testing a loop with counter control
' Adds numbers from 1 to n
' Assigns the value of the calculation to variable x
' Writes the value of x in cell B7

' Extract the value of n

Sheets("Sheet1").Select
Range("B6").Select
n = ActiveCell.Value

' initialize the variable x to keep track of our calculation

x = 0

' start the calculation loop

For i = 1 To n
    x = x + i
Next i

' select cell B7 and write down the calculation of the loop

Range("B7").Select
ActiveCell.Value = x

End Sub
```

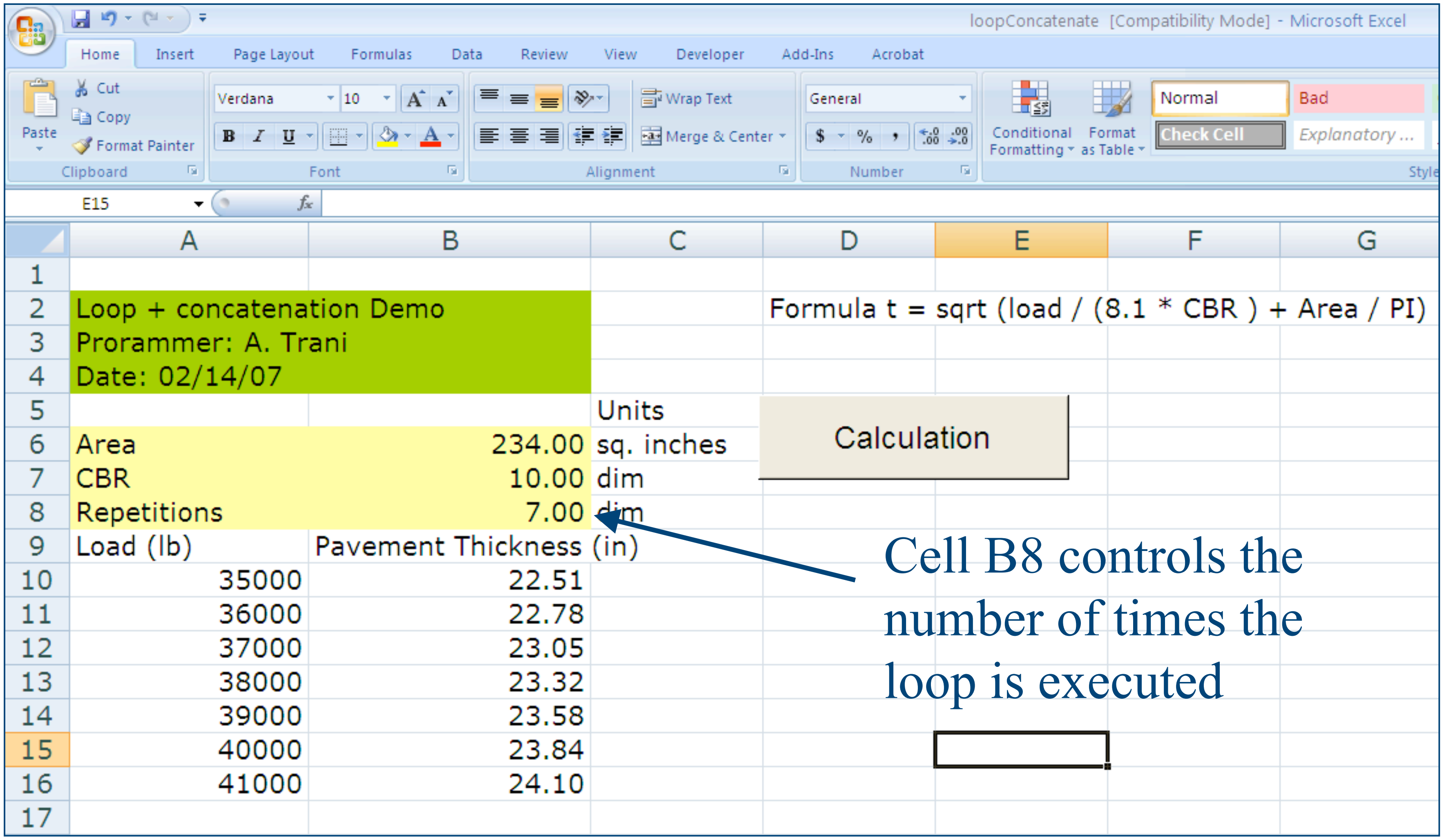
A Loop with Concatenation Control

- The program in worksheet: loopConcatenate.xls

offers a sample of a loop computation and the use of concatenation control to estimate pavement thicknesses

- The pavement thickness function created in previous classes in “called” by the VBA code

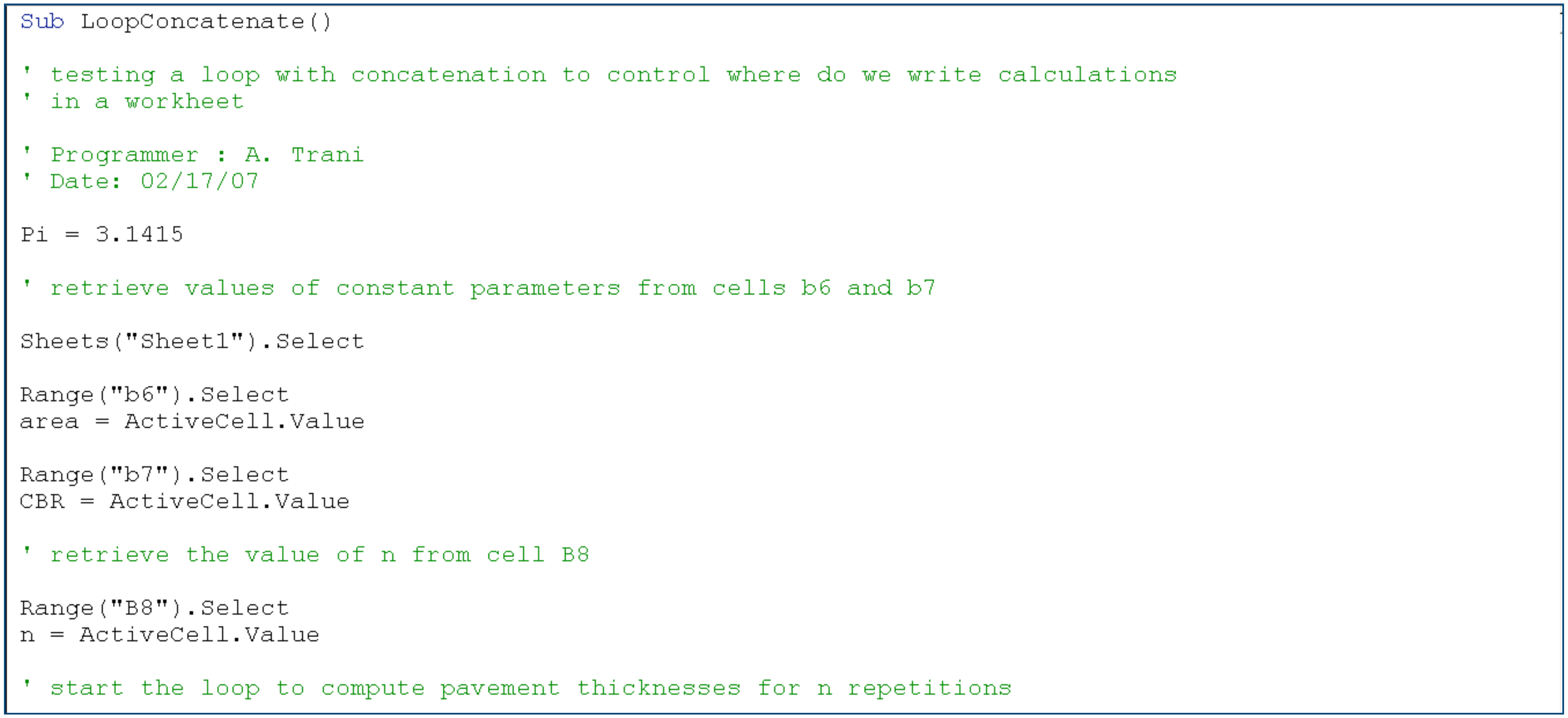
Worksheet Interface



	A	B	C	D	E	F	G
1							
2	Loop + concatenation Demo			Formula t = sqrt (load / (8.1 * CBR) + Area / PI)			
3	Prorammer: A. Trani						
4	Date: 02/14/07						
5			Units				
6	Area	234.00	sq. inches				
7	CBR	10.00	dim				
8	Repetitions	7.00	dim				
9	Load (lb)	Pavement Thickness (in)					
10	35000	22.51					
11	36000	22.78					
12	37000	23.05					
13	38000	23.32					
14	39000	23.58					
15	40000	23.84					
16	41000	24.10					
17							

Cell B8 controls the number of times the loop is executed

The Code Behind the Worksheet



```
Sub LoopConcatenate()

' testing a loop with concatenation to control where do we write calculations
' in a workheet

' Programmer : A. Trani
' Date: 02/17/07

Pi = 3.1415

' retrieve values of constant parameters from cells b6 and b7

Sheets("Sheet1").Select

Range("b6").Select
area = ActiveCell.Value

Range("b7").Select
CBR = ActiveCell.Value

' retrieve the value of n from cell B8

Range("B8").Select
n = ActiveCell.Value

' start the loop to compute pavement thicknesses for n repetitions
```

Code (cont.)

```
' retrieve the value of n from cell B8

Range("B8").Select
n = ActiveCell.Value

' start the loop to compute pavement thicknesses for n repetitions

For i = 1 To n

    cellNumber = "A" & (i + 9)                    ' assign the cell to write load values
    Range(cellNumber).Select                      ' select cell assigned in previous step
    appliedLoad = 35000 + 1000 * (i - 1)          ' compute load (lb) at 1000 lb increments
    ActiveCell.Value = appliedLoad                ' assign computed load to cells A+ (n+9)

    ' calculate the pavement thickness

    thickness = Sqr(appliedLoad / (8.1 * CBR) + area / Pi)

    cellNumber = "B" & (i + 9)                     ' assign the cell to write pavement thickness values
    Range(cellNumber).Select                       ' select cell
    ActiveCell.Value = thickness                   ' write value of pavement thickness

Next i                                            |' next value of i

End Sub
```

Concatenation

Calls Function Thickness

Try Other Refinements

- Currently the loop counter just overwrites the values of pavement thickness without erasing previous computation
- Try adding a line or two of code to erase the previous table of computations while executing the code

VB Basic Coding
With… End With

- The With… End With structure is used to optimize code by speeding up code execution:
  - apply multiple properties and methods to the same object

```
With ActiveCell
      .Clear
      .Value = “Greetings”
      .Font.Bold = True
      .RowHeight = 11
      MsgBox.Address
End With
```

VB Advanced Coding
Manipulating Ranges

- **Return single cell:**

```
Set c = ActiveCell
```

  - points object variable to active cell

```
ActiveSheet.Range("C10").Activate
```

  - activates cell C10

- **Multiple cell ranges:**

```
Worksheets(1).Range("Years")
Worksheets(1).Range("C2:F13").Font.Bold = True
Range(Cells(2,3),Cells(13,6)).Font.Color = vbRed
```

VB Advanced Coding
Row, Column, and Cell Manipulation

- Examples:

```
Worksheets(1).Columns(3).AutoFit
```

  - changes the width of the third column

```
Worksheets(2).Columns(“A:K”).AutoFit
```

  - changes the width of columns A to K to achieve best fit

```
Worksheets(2).Range(Rows(10),Rows(15)).Delete
```

  - deletes rows 10 through 15

```
Worksheets(2).Cells(2,1) = 13
```

  - sets the value of A2 to 13

A Simple Program with a Loop

- Loops are natural ways to execute computations that require multiple iterations
- Loops can be conditioned or controlled by a counter

- Conditional loops - when some condition is used to exit the loop
- Counter controlled loops - when the number of passes in the loop is known