Outline

- Review last class
  - VBE Editor, Variables and Declarations, Arithmetic Operators and Statements
- Logical and Relational Operators
- More on variables and constants
- Program decisions using If statements
- Conditional looping (Do)
- Count-controlled looping (For – Next)
- Work on programming assignment

Review Declaring Variables

- Use Option Explicit to force declarations
  - Common data types: long, double, string, Boolean and date
  - Variant data type can handle any type
    - Is default for variables not assigned a type
- Examples of Dim statements used to declare variables
  ```vba
  Dim s As String  'Comment here
  Dim y As Double, z As Double
  Dim x, y As Double  'Do not use
  ```

Review Relational Operators

- Program logic requires choices based on expressions that are true or false
- Relational operators compare other variables and have true or false results
- The operators are <, <=, =, >, >=, <>
  - Usual definitions with <> as not equal
    - Literally < is less than or greater than
  - Use \( \text{abs}(x - y) \leq \text{eps} \times (\text{abs}(x) + \text{abs}(y)) \) for equality test on real variables

Logical Operators

- Combine true/false values
- Operators are Not, And, Or
- \( x \text{ And } y \) is true if both \( x \) and \( y \) are true
- \( x \text{ Or } y \) is true if either \( x \) or \( y \) is true
- Not \( x \) is true if \( x \) is false and is false if \( x \) is true (Not is like a unary –)
- Is the following true or false?
  - \((6 < 3) \text{ Or } ((12 > 2) \text{ And } (\text{Not}(6<6)))\)
  - \(6 < 3 \text{ Or } 12 > 2 \text{ And } \text{Not } 6<6\)
Operators in Precedence Order

Parentheses ( Evaluate me first )

Exponentiation ^

Unary minus  – (E.g. –x)

Multiply/Divide  * /

Integer Division \

Remainder Mod   (E.g.  7 Mod 4 = 3)

Addition/Subtraction  + –

String concatenation &

Relational  < <= = >= > <=

Logical in following order: Not And Or

Exercise

• A year is a leap year if
  – It is evenly divisible by 4 and not evenly divisible by 100
  – Or it is evenly divisible by 400
• Y is evenly divisible by 4 if Y Mod 4 = 0

Write an expression that is true if a year is a leap year

(year Mod 4 = 0) and not(year mod 100 = 0) or(year mod 400 = 0)

Variable Scope, Persistence

• Scope is where a variable is defined
• Persistence is how long it is defined
• Variables declared in a procedure have procedure scope and persistence
  – They are defined only for that procedure
  – They are reinitialized each time the procedure is called
  – They are called local variables
• Use Static instead of Dim for persistence in a procedure variable between calls

Examples

Sub test()
Dim count as Long
count = count + 1
End Sub

Sub test2()
Static count as Long
count = count + 1
End Sub

Each time sub test is called the variable count is set to zero
All variables are initialized to zero

Scope and Persistence II

• Variables declared in declarations section of module (before first procedure) exist for all procedures in module
• For a workbook with multiple modules
  – Module variables in declarations section with Dim or Private have scope of module
  – Variables set as Public in declarations have scope of all modules are global variables
  – Do not use module or global variables in your programs for this course

Another Example

Option Explicit
Dim k as Long
Private x as Double
Public d as Date
Sub First()
Call Second()
MsgBox "k = " & k & vbCrLf & "x = " & x & vbCrLf & "d = " & d
End Sub

Sub Second()
k = 2
x = 4 * atn(1)
d = Now()
End Sub

Note use of _ for multiline statements
Type Conversion

- Last slide showed example of type conversion done automatically
  - “k = ” & k converted numeric k to string
- When mixed types are used VBA tries to determine the conversion desired by the user and does it
- You can use a set of functions that do the conversions: CDbl(var), CLng(var), CStr(var), CInt(var), CBool(var)

Defined Constants in Office

- Symbolic constants from macro recorder for Office (prefix mso), Excel (prefix xl) and VBA (prefix vb)
- These usually define options which are normally simple integers
- Giving the constants a name makes them easier to remember
- If you want blue, you can use color = vbBlue; you don’t need the blue number

Comments and Colors

- Comments may be placed in the code on a new line or at the end of a completed line, not after a continuation
  - Start with an apostrophe
  - Default color is green in VBE
- Keywords (Function, If, As, …) are blue
  - Editor will convert letters to upper case
  - Will change colors or do case conversion if spelling is not correct

Choice Statements

- The If statement
  If <condition> Then
  <statements to be executed if the condition is true>
  End If
  <Transfer control to first statement after the End If>
  if <condition> is false>
- Alternative version for one statement in If
  If <condition> Then <statement1>
- <condition> must have a Boolean value of true or false

If – Else If

If <condition1> Then
  <Statements done if condition1 is true>
ElseIf <condition2> Then
  <Statements done if condition2 is true>
ElseIf <condition3> Then
  <Statements done if condition3 is true>
  <May be other conditions>
Else
  <Statements done if all conditions false>
End If
  <Execute here after any statements done>

If – Else If Explained

- If any condition is true, the statements following the If or Else If are executed
- Once any statements are executed transfer to the first statement after the End If
- Statements for only the first true condition are executed
- The Else block is optional
  - If no conditions are true those statements are executed
  - If any conditions are false those statements are executed
Use of If Statements

- Consider the following function \( y(x) \)
  - \( y = 0 \) for \( x < 0 \)
  - \( y = x \) for \( 0 \leq x < 1 \)
  - \( y = x^2 \) for \( 1 \leq x \leq 10 \)
  - \( y = 100 \) for \( x > 10 \)
- What is the code to compute \( y \) for any \( x \)?
  ```vba
  If x < 0 then
    y = 0
  ElseIf x < 1 then
    y = x
  ElseIf x <= 10 then
    y = x^2
  ElseIf x > 10
    y = 100
  End If
  ```

Looping

- Count control loop repeats code a fixed number of times
- Conditional looping repeats while a condition is true or until a condition is false
- Both types of loops may be nested
- May use Exit For or Exit Do statements to exit loop before normal exit

Count Controlled Loop

- For loop executes repeatedly with the \(<\text{counter}>\) increasing by the \(<\text{increment}>\) each time
  - Value of \(<\text{counter}>\) may be used in loop
- Loop not executed if \(<\text{counter}>\) becomes greater than \(<\text{end}>\) so loop is executed \(n\text{Times}\) times
  - \(n\text{Times} = (<\text{end}> - <\text{start}>) / <\text{increment}> + 1\)
- Loop not executed if \(n\text{Times} <= 0\)

Count Controlled Examples

- Code at left computes \( e^x \) for \( x = 1 \) with relative error of \(1 \times 10^{-8}\)
  - \( e^x = 1 + x + x^2 / 2! + x^3 / 3! + \ldots \)
Nested For Loops

- Focus on the loops and ignore $x(k) = \text{statements that use arrays, which we haven’t covered yet}$
  
  For $k = n \text{ To } 1 \text{ Step } -1$
  
  $x(k) = a(n,n+1)$
  
  For $j = k+1 \text{ to } n$
  
  $x(k) = x(k) - a(k,j) \times x(j)$
  
  Next $j$
  
  Next $k$

Conditional Loop

- `<cond>` is a condition (can be true or false)
- `<stmts>` are statements executed in the loop (which should change the condition)

<table>
<thead>
<tr>
<th>Do</th>
<th>Do While <code>&lt;cond&gt;</code></th>
<th>Do Until <code>&lt;cond&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;stmts&gt;</code></td>
<td>Loop</td>
<td><code>&lt;stmts&gt;</code> Loop</td>
</tr>
<tr>
<td>if <code>&lt;cond&gt;</code> _ Then Exit Do</td>
<td><code>&lt;stmts&gt;</code> Loop</td>
<td><code>&lt;stmts&gt;</code> Loop While <code>&lt;cond&gt;</code></td>
</tr>
</tbody>
</table>

Looping Exercises

- Open workbook for Looping Exercise from What’s New section of web site
  
  - Uses Taylor series to compute $\sin(x)$
  
  - Enter different $x$ values and observe results
  
  - View code for mySine and mySine2 functions
    
    - mySine uses fixed number of terms
    
    - mySine2 uses fixed allowable relative error
    
    - Modify functions to allow user inputs for
      
      - Total number of terms in mySine
        
        - Relative error and maximum iterations in mySine2

Looping Exercises II

- What you will have to do
  
  - Modify function headers to introduce new variables for number of terms in mySine and allowed error and maximum allowed iterations in mySine2
  
  - Modify code to use values of these variables in place of current constant values for these three terms
  
  - Change function calls in worksheet to allow user to enter values for the new variables

Looping Exercises III

- Insert rows for variables for iterations in mySine and allowed error and maximum allowed iterations in mySine2 and edit formulas

mySine Code

```vba
mySine = x
term = x
For k = 1 To 4
    term = -term * x ^ 2 / _
          ((2 * k + 1) * (2 * k))
    mySine = mySine + term
Next k
```

For loop limit is number of terms minus one
VBA Review Continued

### mySine2 Code

```vba
mySine2 = x
term = x
k = 1
Do
    oldTerm = term
    term = -term * x ^ 2 / ((2 * k + 1) * (2 * k))
    mySine2 = mySine2 + term
    Converged = Abs(term - oldTerm) <= 0.0001 * Abs(mySine2)
    k = k + 1
Loop Until Converged Or k > 100
```

---

### Tonight’s Tasks

- Open Excel workbook from [assignment](assignment) link on home page of course web site
  - Contains worksheets for first two parts of assignment and the VBA code from the assignment document (do not have to copy)
    - Can also open document for first programming assignment from same web location
  - On the Cells tab complete the cell entries as described at the end of the last class
    - See next slide

### First Programming Assignment

- Compute x(t) and y(t) from t = 0 to t = t_max for a trajectory for given V_0 and θ
  \[ y = V_0 t \sin \theta - \frac{g t^2}{2} \]
  \[ x = V_0 t \cos \theta \]
  \[ t_{max} = \frac{2V_0 \sin \theta}{g} \]
- Six different approaches using combinations of cell references (B3), range names, user-defined function, array function and macro
  - Work on first four parts — cell formulas, user defined functions and range names

### Part 2: User-Defined Function

- Here is code given in assignment
  ```vba
  Option Explicit
  Const g As Double = 9.80665
  Const degreesToRadians As Double = 3.14159265358979 / 180
  Function maxTime(ByVal v0 As Double, ByVal theta As Double) As Double
      maxTime = 2 * v0 * Sin(theta * degreesToRadians) / g
  End Function
  ```
- To use function in spreadsheet
  ```excel
  =B5/B4
  =B10+$B$6
  ```

### Part 2: User-Defined Function II

- Use maxTime code as example to write UDFs for x and y
  ```vba
  Function x(ByVal v0 As Double, ByVal theta As Double) As Double
      x = V0 * Cos(theta * degreesToRadians)
  End Function
  ```
  ```excel
  =B5/B4
  ```
- What arguments are required for x and y?
- What are statements to compute x and y?
- How do you call functions from worksheet?
More on First Program

- Array expressions: formulas that are entered into multiple cells at the same time
- Array functions: functions whose values are entered into multiple cells as arrays
- When entering an array formula or using an array function you must press Control+Shift+Enter instead of Enter
- You are asked to modify VBA code to use an array formula for the trajectory

Use of Macro

- User enters data then clicks button to get trajectory
- Macro that button executes is given on exercise
  - Uses function calculateTrajectory used in previous part of assignment

Result of Array Formula Exercise

Instructions:
Enter values for initial velocity in m/s in cell B1 and initial angle in degrees in cell B2.
Select a range of 3 columns with the desired number of rows to display the trajectory plus the header.
With the full range still selected type one of the following formulas in the formula bar:
=trajectory(B1,B2)
=trajectory(Initial_velocity, Initial_angle)
Press control+shift+enter to enter the array formula.

Function trajectory(v0 As Double, theta As Double) As Variant

Dim userRows As Long
Dim userColumns As Long
'Statements below determine rows and columns
userRows = Application.Caller.Rows.Count
userColumns = Application.Caller.Columns.Count
If userColumns < 3 Then
  'Removed code with error message
Else
  trajectory = calculateTrajectory(v0, theta, userRows)
End If
End Function

Sub getTrajectory()

‘Comments and declarations not shown here
v0 = Range("b1").Value
theta = Range("b2").Value
nRows = Range("b3").Value + 2
Cells(startRow, startCol) _
  .CurrentRegion.ClearContents
Range(Cells(startRow, startCol),
  Cells(startRow + nRows - 1, startCol + 2)) _
  .Value = calculateTrajectory _
    (v0, theta, nRows)
End Sub
Note to LSC

• The remaining slides were present in the file from Fall 2012, but were not used in the Spring 2014 Presentation. Only slides 1-42 were placed in the online presentation.

Choice Function

• Write and test a VBA function that takes a value from a worksheet and returns an index (to the worksheet) based on the value ranges shown in the table below:

<table>
<thead>
<tr>
<th>Values</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>x &lt; 0</td>
<td>Err Low</td>
</tr>
<tr>
<td>0 ≤ x &lt; 0.1</td>
<td>1</td>
</tr>
<tr>
<td>0.1 ≤ x &lt; 0.5</td>
<td>2</td>
</tr>
<tr>
<td>0.5 ≤ x &lt; 1</td>
<td>3</td>
</tr>
<tr>
<td>1 ≤ x &lt; 2</td>
<td>4</td>
</tr>
<tr>
<td>2 ≤ x &lt; 5</td>
<td>5</td>
</tr>
<tr>
<td>5 ≤ x ≤ 10</td>
<td>6</td>
</tr>
<tr>
<td>x &gt; 10</td>
<td>Err High</td>
</tr>
</tbody>
</table>

Solution to Exercise

Function label(x As Double) As Variant
If x < 0 Then
    label = "Value too low"
ElseIf x < 0.1 Then
    label = 1
ElseIf x < 0.5 Then
    label = 2
ElseIf x < 1 Then
    label = 3
ElseIf x < 2 Then
    label = 4
ElseIf x < 5 Then
    label = 5
ElseIf x <= 10 Then
    label = 6
Else
    label = "Value too high"
End If
End Function

Solution to Exercise II

ElseIf x < 5 Then
    label = 5
ElseIf x <= 10 Then
    label = 6
Else
    label = "Value too high"
End If

Select Case

• Lists possible values (or ranges of values) called cases for a variable
Select Case <variable or expression>
    Case <specification>
        <statements to be executed if case meets specification>
    <Repeat other cases including Case Else>
End Select

Only first case found to be true is executed

Select Case Example

Select Case x
    Case a
        <statements to be executed if x = a>
    Case b to c
        <statements executed if b ≤ x ≤ c>
    Case d, e, f to g
        <executed if x = d, x = e or f ≤ x ≤ g>
    Case Is >= h
        <statements executed if x >= h>
    Case Else
        <executed if all other cases false>
End Select
Redo Problem with Select Case

- Get an index based on the value ranges shown in the table below
- Problem is inclusive nature of Case a to b
- Solution is to write cases in reverse order

<table>
<thead>
<tr>
<th>Values</th>
<th>Index</th>
<th>Values</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x &lt; 0 )</td>
<td>Err Low</td>
<td>( 1 \leq x &lt; 2 )</td>
<td>4</td>
</tr>
<tr>
<td>( 0 \leq x &lt; 0.1 )</td>
<td>1</td>
<td>( 2 \leq x &lt; 5 )</td>
<td>5</td>
</tr>
<tr>
<td>( 0.1 \leq x &lt; 0.5 )</td>
<td>2</td>
<td>( 5 \leq x \leq 10 )</td>
<td>6</td>
</tr>
<tr>
<td>( 0.5 \leq x &lt; 1 )</td>
<td>3</td>
<td>( x &gt; 10 )</td>
<td>Err High</td>
</tr>
</tbody>
</table>

Select Case Solution

Function label(x As Double) As Variant

Select Case x
Case Is > 10
    label = "Value too high"
Case 5 To 10
    label = 6
Case 2 To 5
    label = 5
Case 1 To 2
    label = 4
Case 0.5 To 1
    label = 3
End Select
End Function

Excel Function Example

- Compute loan payment using PMT
- PMT(Rate, Periods, LoanAmount)
- Sample of PMT
- How does PMT know B1 is rate?
- PMT uses the cell values in the function call as rate, periods, amount to find PMT
- When using a function arguments to function must be in correct order

From PMT to Your UDF

- Whoever wrote the PMT function decided on the order of the arguments
- Similarly when you write a UDF, you decide on the order of the arguments
- Anyone who uses your function must follow the order you set
- The function may be called from a worksheet or other VBA procedure
  - But the order must be correct

Writing Your UDF

- The header has the following form
  Function <name>(<arguments>) As <type>
  - <name> is the name of the function
  - <type> is the data type for the function
  - <arguments> may be blank or have one or more entries of the form <variable> As <type>
    - <variable> is a variable used in the function
    - <type> is the data type for that variable
    - Separate multiple entries in the list by commas
  - Arguments provide input data to function

ME 309 – Numerical Analysis of Engineering Systems
Writing Your UDF II

- The function has the following form
  Function <name> ( <arguments> ) As <type>
  <code to do computations>
  <name> = <value from computations>
End Function

Function vCyl ( R as Double, H as Double ) As Double
  vCyl = 4 * atn(1) * R^2 * H
End Function

Using Your UDF

- Use by cell entries in worksheet
  =vCyl( B1, B2)
  =vCyl( 1, 20)
  =vCyl( radiusName, heightName)

- Call from other VBA procedures
  V = vCyl( radius, height)
  cylVol = vCyl( 1, 20)
  v10cyls = 10 * vCyl( rad, hgt)

Arrays

- Arrays can be visualized as data on an experimental variable
  - Could describe pressure data points mathematically as \( P_1, P_2, \ldots \).
  - In VBA we can represent these data points as \( P(1), P(2), \ldots \).
  - We call the numbers (1, 2, etc.) indices or subscripts.
  - We can use constants or variables for the subscripts: \( P(4), P(k), \) where \( k \) has a value.

Two-dimensional Arrays

Consider an experiment where you vary the current over six levels, the voltage over four levels and measure the efficiency, \( e \), of an electromechanical device. The data for each combination of current and voltage can be represented as shown below.

<table>
<thead>
<tr>
<th>( l(1) )</th>
<th>( l(2) )</th>
<th>( l(3) )</th>
<th>( l(4) )</th>
<th>( l(5) )</th>
<th>( l(6) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V(1) )</td>
<td>( e(1,1) )</td>
<td>( e(1,2) )</td>
<td>( e(1,3) )</td>
<td>( e(1,4) )</td>
<td>( e(1,5) )</td>
</tr>
<tr>
<td>( V(2) )</td>
<td>( e(2,1) )</td>
<td>( e(2,2) )</td>
<td>( e(2,3) )</td>
<td>( e(2,4) )</td>
<td>( e(2,5) )</td>
</tr>
<tr>
<td>( V(3) )</td>
<td>( e(3,1) )</td>
<td>( e(3,2) )</td>
<td>( e(3,3) )</td>
<td>( e(3,4) )</td>
<td>( e(3,5) )</td>
</tr>
<tr>
<td>( V(4) )</td>
<td>( e(4,1) )</td>
<td>( e(4,2) )</td>
<td>( e(4,3) )</td>
<td>( e(4,4) )</td>
<td>( e(4,5) )</td>
</tr>
</tbody>
</table>

Declaring Arrays

- Arrays must be declared as arrays
- The maximum size of the array must be specified in the Dim statement
- In VBA the lowest array subscript is zero by default
  - Can use Option Base 1 in declarations section to change default lowest subscript to one
  - Can also set lowest subscript on each individual array

Dimensioning Arrays

- Can declare arrays as follows
  Dim I(1 to 6) as double
  Dim V(1 to 4) as double
  Dim e(1 to 4, 1 to 6) as double
- Size below depends on Option Base
  Dim I(6) as double
  Dim V(4) as double
  Dim e(4, 6) as double

What is lowest subscript for these arrays?
Zero or one depending on Option Base
Using Arrays

- Arrays components are referenced by their subscripts
- This is often done in a For loop
  \[ x(k) = \sin(k \times \pi / 100) \]  
  \( \pi = 3.14 \ldots \)
  \( \text{Next } k \)
- \( x \) is an array with 101 components giving \( \sin(x) \) for \( 0 \leq x \leq \pi \), with \( \Delta x = \pi / 100 \)

Two-Dimensional Arrays

- Use nested for loops
  - Use example of current and voltages
    \[ \text{For } k = 1 \text{ to } 4 \]
    \[ \text{For } j = 1 \text{ to } 6 \]
    \[ \text{Power}(k,j) = I(j) \times V(k) \]
    \( \text{Next } j \)
    \( \text{Next } k \)
- Recall table:
  - \( V \) was in rows
  - \( I \) was in columns
  - \( \text{Power}(k,j) \) is \( \text{Power} \) (row, column)
- Are \( k \) and \( j \) indices correct?

Dynamic Arrays

- What if you do not know array size until program is actually running?
- Use Dim a() to tell compiler that a is an array then use ReDim with actual dimensions

Passing Arrays to Procedures

- Declare array in argument list with parentheses to indicate array
  \[ \text{Sub mine}( A() \text{ as double}) \]
  *No dim statement for A*
  \[ A(2,3) = \]
- Calling program sets actual dimensions on array and uses only the following
  \[ \text{Dim } B(1 \text{ to } 10, \text{ 1 to } 6) \text{ as double} \]
  \[ \text{Call mine}(B) \]

Determining Array Bounds

- The UBound and LBound functions determine the upper and lower bounds of unknown array dimensions
- For a two-dimensional array, \( A(m,k) \)
  - \( \text{LBound}(A,1) \) is the lower bound of \( m \)
  - \( \text{UBound}(A,1) \) is the upper bound of \( m \)
  - \( \text{LBound}(A,2) \) is the lower bound of \( k \)
  - \( \text{UBound}(A,2) \) is the upper bound of \( k \)

Worksheet Arrays to VBA

- Passed as a range of cells
- First step is to set a type variant variable equal to the input range variable
  - The variant variable is now an two-dimensional array
  - May have single row or single column, but is still a two-dimensional array
  - Lower bound is always one
  - Can use UBound to get sizes
Worksheet Array Example

Function getMean (Ain As Range) As Double
Dim A as Variant, m as Long, k as Long
Dim sum as double, cells as Long
Dim nRows as Long, nCols as Long
A = Ain:
nRows = UBound(A,1)
nCols = UBound(A,2) : cells = nRows * nCols
For k = 1 to nRows
  For m = 1 to nCols
    sum = sum + A(k,m)
  Next m
Next k
getMean = sum / cells
End Function

Worksheet Array Example II

Dim sum as double, cells as Long
Dim nRows as Long, nCols as Long
A = Ain:
nRows = UBound(A,1)
nCols = UBound(A,2) : cells = nRows * nCols
For k = 1 to nRows
  For m = 1 to nCols
    sum = sum + A(k,m)
  Next m
getMean = sum / cells
End Function

VBA Array to Worksheet

• VBA steps to return array to worksheet
  – Declare the function type as Variant
  – In the function or sub declare a working
    array for calculations
    • Use application.caller for dimensions
    • Write the code for values in working array
    – At end of function set <function name> =
      <working array name>
  • To use the function: select cells; enter
    function in formula bar; Cont+Shift+Enter

Passing by Reference/Value

• Consider the following function call
  Call mySub( a, b)
Sub mySub( x as Long, y as Long)
  x = 2 * x
  y = x / y
End Sub
• What happens to the value of a in the
  calling program because of the x = 2 * x?
  Answer: The value of a will have the new value
  of x computed in myFunc

Passing by Reference/Value 2

• By default VBA passes memory
  locations of variables to procedures
  – This is known as pass by reference
• Alternative is pass by value
  – This simply sends the procedure the value
    stored in the memory location
  – To use pass by value enter the keyword
    ByVal before the variable in the header
Sub mySub( ByVal x as Long, y as Long)
Strings

• Consider only variable length
• Use Dim str as String
• Use & or + as concatenation operator to join two strings
• Len(str) gives length of string
• Left, Right, and Mid give substrings in same manner as worksheet functions
• InStr function searches for substrings

Debugging

• Debugger allows you to step through a program and see intermediate values
  – Useful to find location of errors
• Items to use in debugger
  – Breakpoints stop execution at certain points
  – Step-by-step execution
  – Intermediate and Watch windows
  – Hover mouse over variable to get its value
  – Change statement to be executed next

Help

• Help systems for Excel and VBA
• Search function does not always return what you are looking for
• If you know the keyword, type it, place the cursor in the keyword, and press F1
• Sometimes a Google search for “VBA <subjectYouAreInterestedIn>” works