GNG 1106
Fundamentals of Engineering Computation
Introduction to Visual Basic

Gilbert Arbez

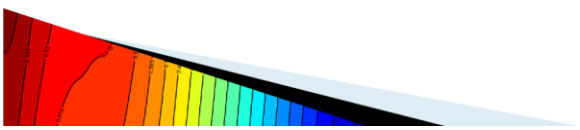
Winter 2016

Introduction to Visual Basic

What is Visual Basic?

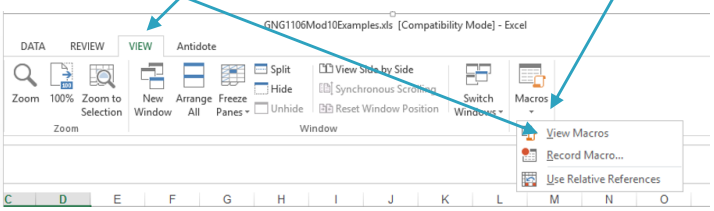
- **Visual Basic** is a tool that allows you to develop GUI applications for Windows (Graphic User Interface – GUI).
- Applications like Excel include Visual Basic for Applications (VBA)
- Visual Basic is event-driven, meaning code executes in response to some event (button pressing, menu selection, ...).
- Some Features of Visual Basic
  - Full set of objects – you 'draw' the application's interface
  - Lots of icons and pictures for your use
  - Response to mouse and keyboard actions
  - Clipboard and printer access
  - Many mathematical, string handling, and graphics functions
  - File support
  - Useful debugger and error-handling facilities
  - Powerful database access tools
  - ActiveX support
  - Package & Deployment Wizard makes distributing your applications simple

Topic 1: Running Visual Basic in Excel

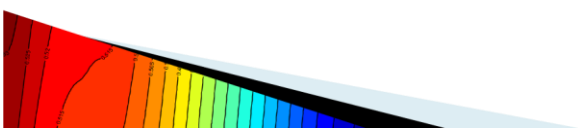


Accessing a macro in Excel

- In Excel, select the tab *VIEW*, then open the menu *Macros*, then *View Macros*
- A *Macro* window shall appear (see the next slide)

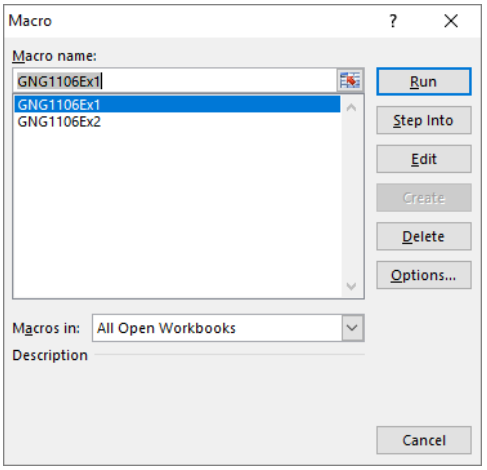


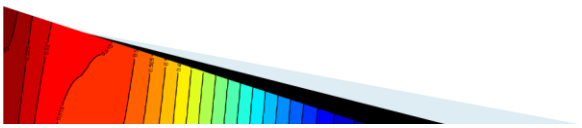
- Note: Be sure to enable the macros when you open the Excel file.



Editing Existing Macro

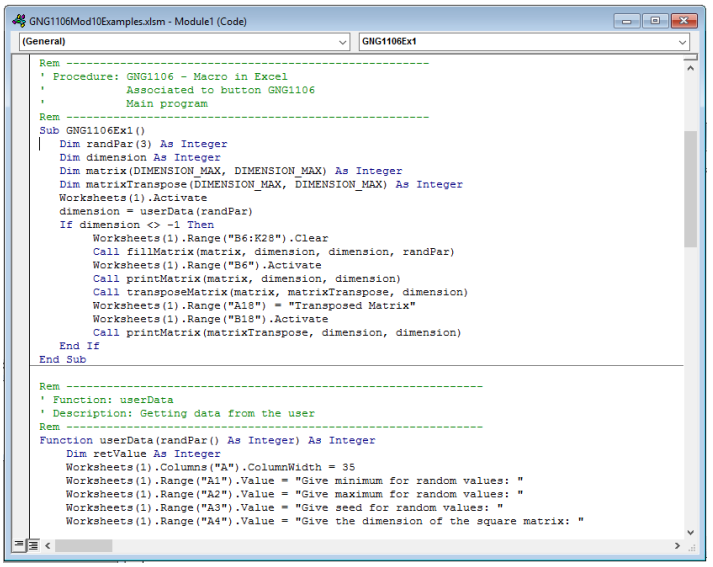
- To edit an existing macro
  - Click on the macro name in the list
  - Click on *Edit*
  - A *Visual Basic* interface shall appear (see next slide).

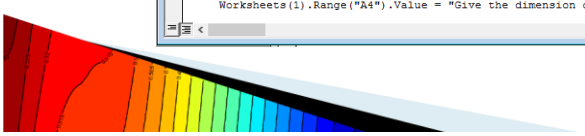




5

Macro is a Visual Basic Program

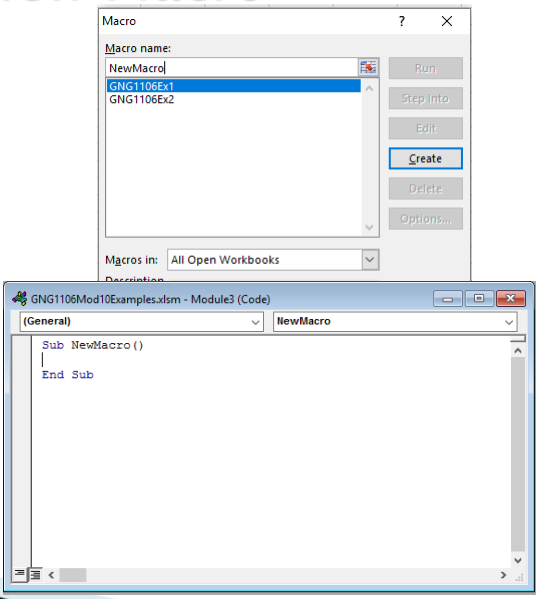




6

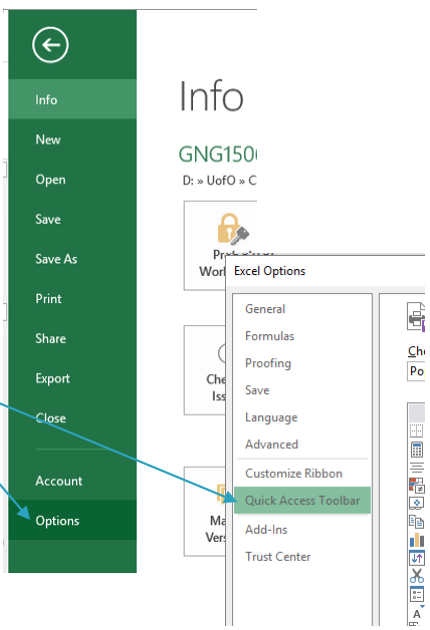
Creating a new Macro

- To create a new macro
  - Type in a new name in the text box under the label *Macro name:*
  - The *Create* button becomes available
  - Click on *Create*
  - A *Visual Basic* interface shall appear with a window that allows you to add code for the new macro.



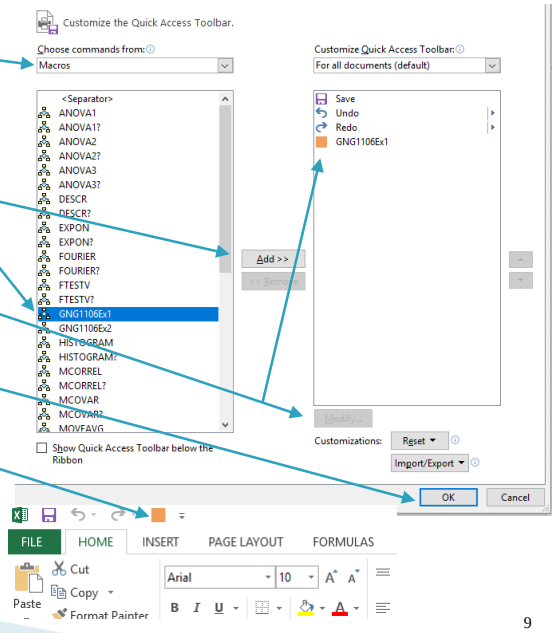
Adding a Button to the Toolbar for the Macro (VB program)

- Select *File* | *Options* and in the window that appears *Quick Access Toolbar*
- The window *Customize the Quick Access Toolbar* shall appear (see next slide).



Adding a Button to the Toolbar for the Macro (VB program)

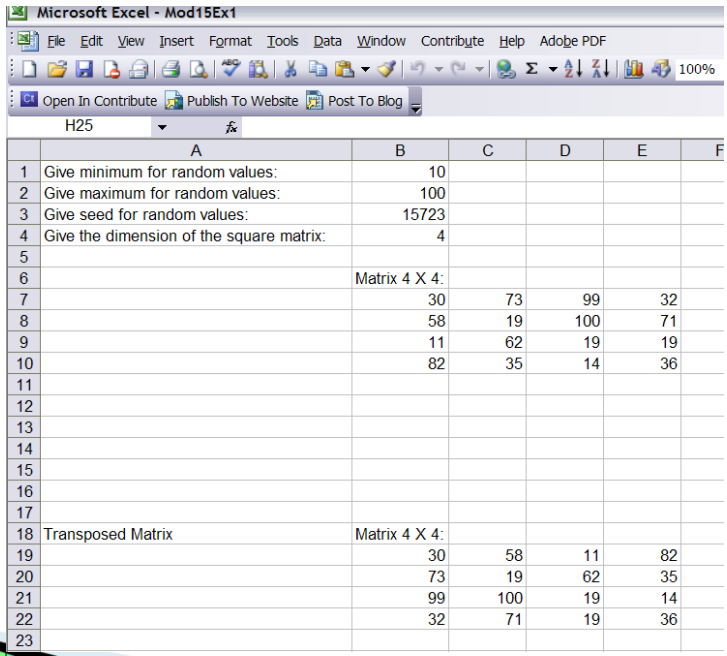
- Select *Macros* in the field *Choose commands from*:
- Select the desired macro
- Click *Add* (the macro name will appear in the right list)
- Click on Modify to select and icon for the macro
- Click OK to close the window.
- An icon appears in the *Quick Access Toolbar*, which allows you to execute the macro.



Example 1 (Transposing a matrix)

- Problem Statement
  - To transpose a square matrix (n X n)
- Gathering of Information and Description of Input/Output
  - Engineering uses matrices, for example, they can be used to analyze the effect of earthquakes on structures (Matrix Analysis of Structural Dynamics: Applications and Earthquake Engineering, Franklin Y. Cheng, CRC Press, 2000)
  - Transposing a matrix consists of moving the values in the rows into the corresponding columns.
  - For example, if $M = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$, then the transposed matrix is $M^T = \begin{matrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{matrix}$
  - Input: Dimension of the square matrix (maximum 10), minimum value, maximum value, and seed for generating random numbers.
  - Output: the matrix and the transposed matrix

Example 1



	A	B	C	D	E	F
1	Give minimum for random values:	10				
2	Give maximum for random values:	100				
3	Give seed for random values:	15723				
4	Give the dimension of the square matrix:	4				
5						
6		Matrix 4 X 4:				
7		30	73	99	32	
8		58	19	100	71	
9		11	62	19	19	
10		82	35	14	36	
11						
12						
13						
14						
15						
16						
17						
18	Transposed Matrix	Matrix 4 X 4:				
19		30	58	11	82	
20		73	19	62	35	
21		99	100	19	14	
22		32	71	19	36	
23						

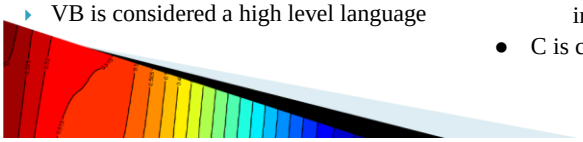
11

Topic 2: Comparing C to Visual Basic

12

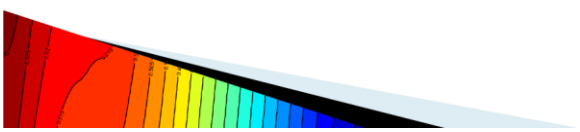
Visual Basic versus C

Visual Basic	C
Variables ◦ Many types ◦ Constant variable	Variables – Few simple types – Symbolic constants
Operators ◦ Arithmetic, comparison, logical	Operators – Arithmetic, comparison, logical
Decision Instructions	Decision Instructions
Looping Instructions	Looping Instructions
Arrays	Arrays
Sub-programs ◦ Procedure ◦ Function	Sub-programs – function with type void – function that returns a value – void function with no parameters
Objects ◦ members include variables and methods (procedures and functions)	Structures – Members are variables and can include pointer to functions
VB is considered a high level language	C is considered a mid-level language



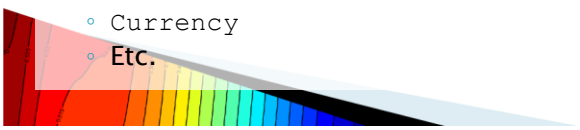
Excel Interface

- Visual Basic treats the Excel WorkSheet an Object
  - There are numerous existing objects when running Excel – such objects can be seen as global structure variables
  - *WorkSheets(1)* references the first worksheet
  - *WorkSheets("Sheet1")* also references the worksheet (the worksheet with name *Sheet1*)
  - *ActiveSheet* references the current active work sheet.
- Cells of a worksheet can be accessed as a member of the Worksheet object
  - *WorkSheets(1).Cells(row,col)*, where row and col are the row and column numbers (indexes that start at 1, NOT at 0, for cells). So
    - *Cells(1,1)* accesses cell A1 in the Excel worksheet
  - *WorkSheets(1).Range("A1"),* can be used to access the cell using its name enclosed in quotes.
  - *WorkSheets(1).Range("B6:K28")* can be use to reference a range of cells.
  - Such expressions can be used to treat the cell as a variable location. Note that many types of values can be assigned to the cell (including strings).
- Given that Excel serves as an interface, what needs to be changed to interact with the user for the example



Variables

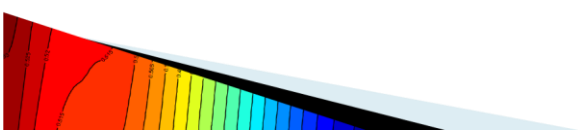
- Computer variables are available in Visual Basic. Rules used in naming variables:
  - No more than 40 characters
  - They may include letters, numbers, and underscore (_)
  - The first character must be a letter
  - You cannot use a reserved word (word needed by Visual Basic)
- Data Types
  - `Boolean` (values True (-1) and False (0) – 2 byte value)
  - `String`
  - `Integer` (2 byte integer number)
  - `Single` (4 byte real number)
  - `Double` (8 byte real number)
  - `Array`
  - `Long` (4 byte number)
  - `Currency`
  - Etc.



15

Variable Declaration

- Within a procedure, variables are declared using the **Dim** statement:
  - **Dim** *VarName* **As** *DataType*
    - **Dim** is the *keyword* that tells Visual Basic that you want to declare a variable
    - *VarName* is the name of the variable
    - **As** is the keyword that tells Visual Basic that you're defining the data type for the variable
    - *DataType* is the data type of the variable
- Global level variables and local variables existent in Visual Basic just like in C.
  - Local variables are declared within procedures and functions
  - Global variables are declared using the keyword **Global**
    - **Global** MyInt **as** Integer
    - **Global** MyDate **as** Date



16

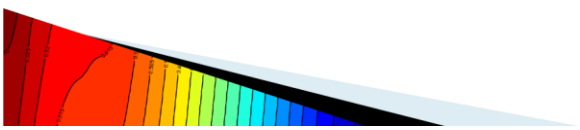
Constants

- Constants, once declared and initialized cannot be changed. The syntax for declaring variables is as follows:

  ```
  Const constName As datatype = value
  ```

- Plays the same role as symbolic constants in C
- Examples:

```
Const SIZE As Integer = 100
Const companyName As String = "GNG1106"
```



Visual Basic Statements and Expressions

- **Assignment** statement

```
StartTime = Now
Explorer.Caption = "Captain Spaulding"
BitCount = ByteCount * 8
Energy = Mass * LIGHTSPEED ^ 2
NetWorth = Assets - Liabilities
```

- Statements normally take up a single line with no terminator. Statements can be **stacked** by using a colon (:) to separate them. Example:

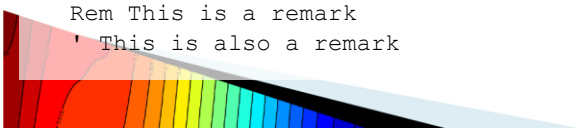
```
StartTime = Now : EndTime = StartTime + 10
```

- If a statement is very long, it may be continued to the next line using the **continuation** character, an underscore (_). Example:

```
Months = Log(Final * IntRate / Deposit + 1) _
  / Log(1 + IntRate)
```

- Comment statements begin with the keyword **Rem** or a single quote ('). For example:

```
Rem This is a remark
' This is also a remark
```



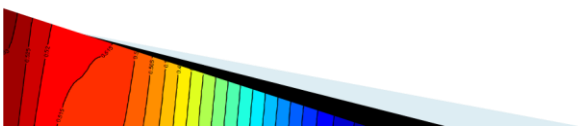
Visual Basic Operators

- The simplest **operators** carry out **arithmetic** operations. These operators in their order of precedence are:

Operator	Operation
^	Exponentiation
* /	Multiplication and division
\	Integer division (truncates)
Mod	Modulus
+ -	Addition and subtraction

- There are six **comparison** operators in Visual Basic:

Operator	Comparison
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
=	Equal to
<>	Not equal to



19

Visual Basic Operators

- We will use three **logical** operators:

Operator	Operation
Not	Logical not
And	Logical and
Or	Logical or

- **Rnd Function** (returns random value between 0 and 1)
  - To produce random integers (I) between Imin and Imax, use the formula:

```
I = Int((Imax - Imin + 1) * Rnd) + Imin
```

  - The random number generator in Visual Basic must be seeded. A **Seed** value initializes the generator.

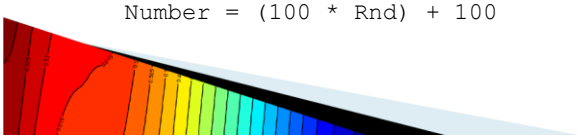
```
Rnd(1234)  Rem The seed is 1234
```

- **Examples**:
  - To roll a six-sided die, the number of spots would be computed using:

```
NumberSpots = Int(6 * Rnd) + 1
```

  - To randomly choose a real number between 100 and 200, use:

```
Number = (100 * Rnd) + 100
```



20

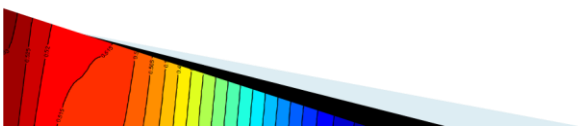
If Statements (same as the C if statement)

- If statements are always in this format:

```
If logical expression Then
    Instruction bloc
End If
```

- Example

```
1  Dim bonus as Decimal = 0
2  Dim sales as Decimal = Worksheets(1).Range("B5");
3
4  If sales > 50000 Then
5     bonus = 100
6  End If
7
8  Worksheets(1).Range("B6"); = "Your bonus is " & _
9                               bonus
```



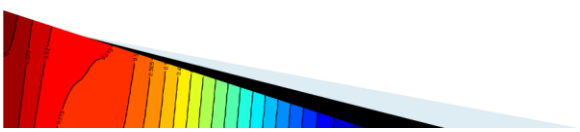
If/Else Statements (same as the C if/else statement)

- If statements are always in this format:

```
If logical expression Then
        Instruction bloc
Else
   Instruction bloc
End If
```

- Example

```
If age >= 26 Then
    message = "You are old …"
Else
    message = "You are young …"
End If
```



If/Elseif Statements

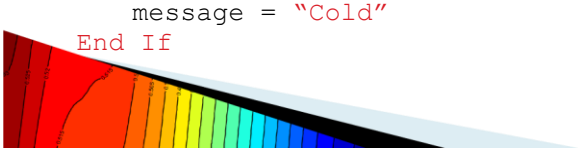
- If statements are always in this format:

```
If logical expression Then
   Instruction bloc
ElseIf logical expression Then
   Instruction bloc
ElseIf …

End If
```

- Example

```
If temperature >= 35 Then
    message = "Too hot …"
ElseIf temperature >= 20 Then
    message = "Moderate Temperature…"
ElseIf temperature >= 0 Then
    message = "Cold"
End If
```



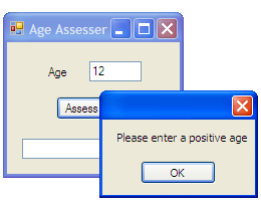
23

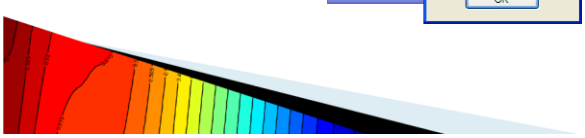
Message Boxes

- Message boxes are useful for telling the user important information
- For example, the line of code …

```
MsgBox(("Hello " & YourName) & ", My friend.")
```

- … will pop-up a window like this, with the appropriate message in it:





24

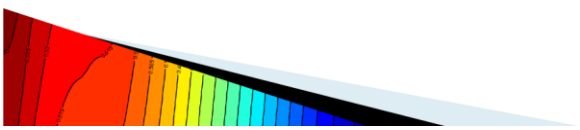
While Loop (same as the C while)

```
While logical expression
    instruction bloc
End While
```

Example

```
While number > 6
    number = Number - 1
    counter = counter + 1
End While

MsgBox("The loop ran " & counter & " times.")
```



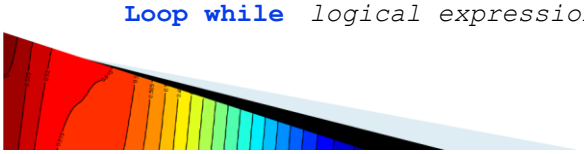
Loops – Do…Loop

- There are two forms of the Do...Loop:
  - Loop is executed as along as the logical expression is True, and terminates when the logical expression becomes False
  - Pre-test version (same as the C while statement)

```
Do While logical expression
  instruction Bloc
Loop
```

  - Post-test version (same as the C do/while statement)

```
Do
        instruction Bloc
Loop while  logical expression
```



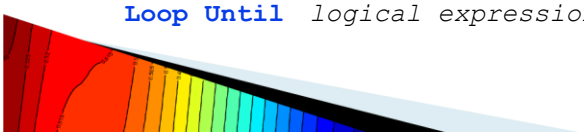
Loops – Do…Until (no C equivalent)

- There are two forms of the Do...Until statement:
  - Loop is executed as along as the logical expression is False, and terminates when the logical expression becomes True, that is repeat the loop UNTIL the logical expression becomes True
  - Pre-test version (similar to the C while statement)

```
Do Until logical expression
  instruction Bloc
Loop
```

  - Post-test version (similar to the C do/while statement)

```
Do
        instruction Bloc
Loop Until  logical expression
```

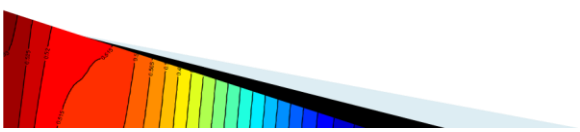


Loops – For...Next Loop (similar to the C `For` loop)

- The Visual basic syntax for the For…Next loop is:

```
For Count = StartValue To EndValue [Step Increment]
    instruction bloc
Next [Count]
```

- *Count* can be any of these types:
  - `Integer` or `Long`
  - `Single` or `Double`
  - `Currency`
  - Using `Integer` type makes the loop run the fastest



For…Next Loop Examples

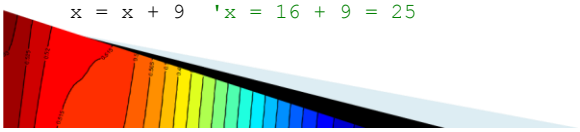
- Example 1:

```
Dim sum as Integer
For i as Integer = 1 to 5
    sum = sum + i
Next
```

- Example 2:

```
Dim x as Integer
For i as Integer = 1 to 10 Step 2
    x = x + i
Next
```

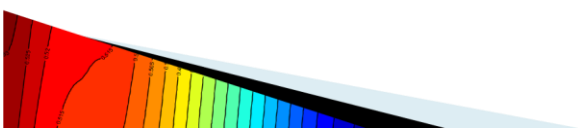
```
x = x + 1  'x = 0 + 1 = 1
x = x + 3  'x = 1 + 3 = 4
x = x + 5  'x = 4 + 5 = 9
x = x + 7  'x = 9 + 7 = 16
x = x + 9  'x = 16 + 9 = 25
```



29

Arrays

- Declaring an Array
  - We use the standard keywords used to declare variables:
    - Dim – local variable
    - Global – global variable
  - Declare number of elements and the types of elements (note that the parenthesis are used in Visual Basic with arrays and not square brackets)
    - `Dim aiCounters(14) As Integer`
      `Dim asNames(5) As String`
      `Static acTicketPrices(5) As Currency`
      `Global gafMeasurements(99) As Single`
  - As in C, the indexes with arrays start at 0
  - It is possible to have indexes start at 1 using the following syntax
    - `Dim aiCounters(1 To 14) As Integer`
      Indexes for `aiCounters` vary from 1 to 14 with the above declaration



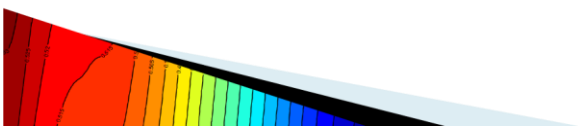
30

Multi-Dimensional Arrays

- Visual basic will allow us up to 60 dimensions!
- What about a three dimensional chess board?

```
Dim asBoard(1 To 8, 1 To 8, 1 To 8) As String
asBoard(1,4,3) = "White Knight"
```

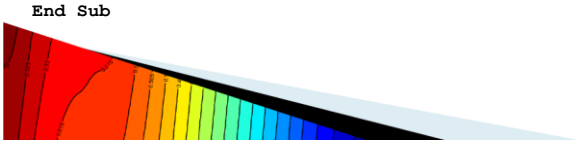
- Best idea is to use multi-dimensional arrays where they map clearly to the real-world



Example of Arrays

- Consider the procedure `fillMatrix` from Example 1
- Change the loops using the For..Next statment.

```
Rem -----------------------------------------------------------------
' Procedure: fillMatrix
' Description: Fills a matrix with random values.
Rem -----------------------------------------------------------------
Sub fillMatrix(mat() As Integer, _
               ByVal nRows As Integer, ByVal nColumns As Integer, _
               randPar() As Integer)
    Dim rix As Integer
    Dim cix As Integer
    Rnd (randPar(SEED)) 'Seeds the random number generator
    rix = 0
    While rix < nRows
       cix = 0
       While cix < nColumns
          mat(rix, cix) = Int((randPar(MAX) - randPar(MIN) + 1) _
                          * Rnd()) + randPar(MIN)
          Rem MsgBox ((((rix & ", ") & cix) & ", ") & mat(rix, cix))
          cix = cix + 1
       End
       rix = rix + 1
    End
End Sub
```



Subprograms in Visual Basic

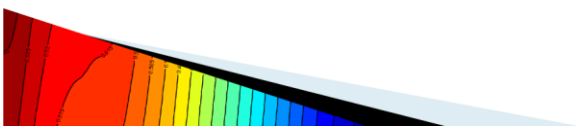
- There are two forms of subprograms in Visual Basic
- The procedure returns no values and is defined as follows:

```
Sub nameProcedure( list of parameters )
    Instruction bloc
End sub
```

- The function must return a value and is defined as follows (note the syntax for defining the type of value returned).

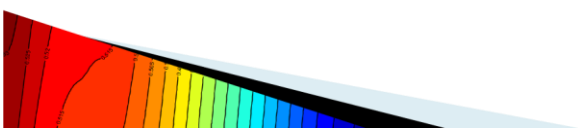
```
Function nameFunction( list de parameters ) As Type
    Instruction Bloc
End Function
```

- List of parameters: pass by reference is used by default with procedures and functions.
  - The keyword ByVal can be used to use pass by value for a parameter
  - The keyword ByRef also exists but is optional
  - See the parameters for the procedure fillMatrix on the preceding slide.



Returning from a procedure and function

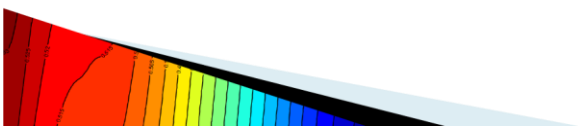
- Returning from a procedure occurs when:
  - The statement `End Sub` is encountered, i.e., at the end of the procedure
  - The statement `Return` or `Exit Sub` is encountered
  - The best style is to NOT use `Return`, nor `Exit Sub` and return from the procedure at its end
- Returning from a procedure occurs when :
  - The instruction `End Function`, is encountered, i.e., at the end of the function
  - The statement `Return` or `Exit Function` is encountered
  - Setting the return value:
    - Assign a value to the function name, as if it were a variable, e.g. for the function `exampleFunc`
      - `exampleFunc = 5`
    - With `Return`, it is possible to return a value, e.g. `Return 5` (note this is NOT available in VBA)
  - The best style is to NOT use `Return`, nor `Exit Return` and return from the function at its end



Function – Example

```
Function CalcSphereVolume( _
  ByVal radius As Decimal) As Decimal
    Dim volume As Decimal = 4 * Math.PI * _
     Math.Pow(radius, 3)
    CalcSphereVolume = volume
 End Function
```

- `Function` – says that this thing is a function
- **`CalcSphereVolume`** – the name of this function
- `ByVal` – parameter radius receives a value passed to it
- **`radius`** `As Decimal` – the name and type of parameter to the function
- `As Decimal` (after the)) – the **`return type`** of this function
- **`CalcSphereVolume = volume`** – sets the return value of the function to **`volume`**
- `End Function` – end the function



Examples of returning values from functions

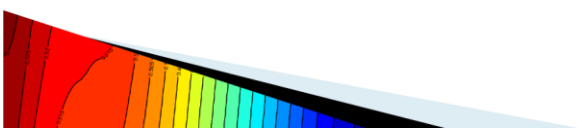
- Recall the two possible ways of returning values from functions:
- Assign the value to be returned to the function name (only option available in VBA)

```
Function myFunction(ByVal j As Integer) As Double
     myFunction = 3.87 * j
End Function
```

OR

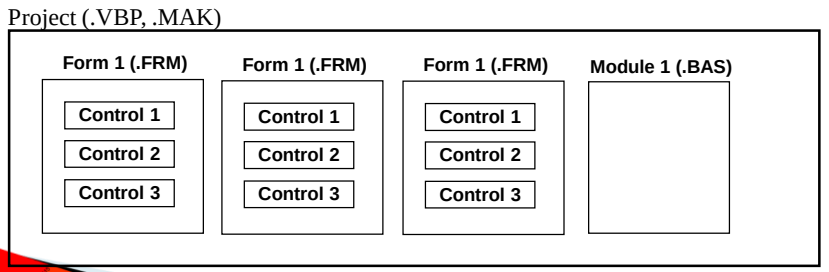
- Include the returned value in a Return statement

```
Function myFunction(ByVal j As Integer) As Double
  Return 3.87 * j
End Function
```



The organization of a Visual Basic Application

- **Application** (Project) is made up of:
  - **Forms** – Windows that you create for user interface
  - **Controls** – Graphical features drawn on forms to allow user interaction (text boxes, labels, scroll bars, command buttons, etc.)
  - **Properties** – Every characteristic of a form or control is specified by a property. Example properties include names, captions, size, color, position, and contents.
  - **Methods** – Built-in procedure that can be invoked to impart some action to a particular object.
  - **Event Procedures** – Code related to some object. It is executed when a certain event occurs.
  - **General Procedures** – Code not related to objects. This code must be invoked by the application.
  - **Modules** – Collection of general procedures, variable declarations, and constant definitions.



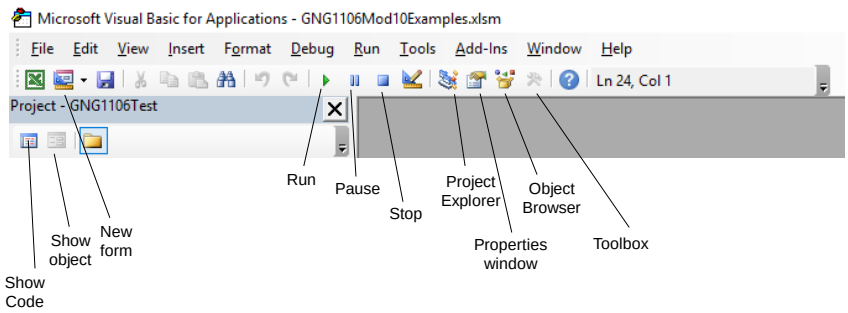
Steps in Developing Application

- There are three primary steps involved in building a Visual Basic application:
  - **Draw** the user **interface**
  - **Assign properties** to controls
  - **Attach code** to controls
- Visual Basic operates in three modes
  - **Design** mode – used to build application
  - **Run** mode – used to run the application
  - **Break** mode – application halted and debugger is available

We focus here on the **design** mode

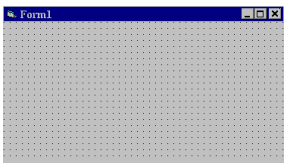
Six windows appear when you start Visual Basic

1. The **Main Window** consists of the title bar, menu bar, and toolbar.

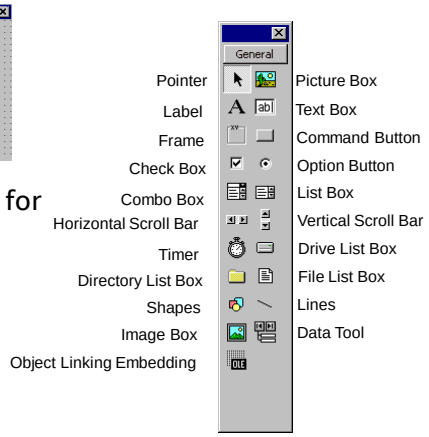


Six windows appear when you start Visual Basic

2. The **Form Window** is central to developing Visual Basic applications. It is where you draw your interface.

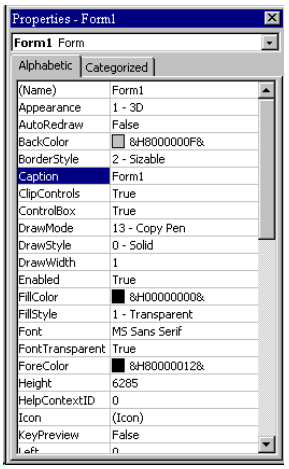


3. The **Toolbox** is the selection menu for controls used in your application.



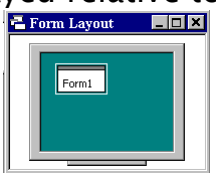
Six windows appear when you start Visual Basic

4. The **Properties Window** is used to establish initial property values for objects. The drop-down box at the top of the window lists all objects in the current form. Two views are available: Alphabetic and Categorized. Under this box are the available properties for the currently selected object.

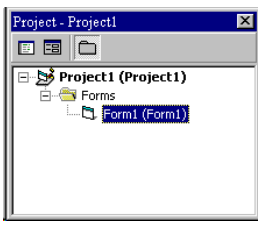


Six windows appear when you start Visual Basic

- The **Form Layout Window** shows where (upon program execution) your form will be displayed relative to your monitor's screen



- The **Project Window** displays a list of all forms and modules making up your application. You can also obtain a view of the **Form** or **Code** windows (window containing the actual Basic coding) from the Project window.

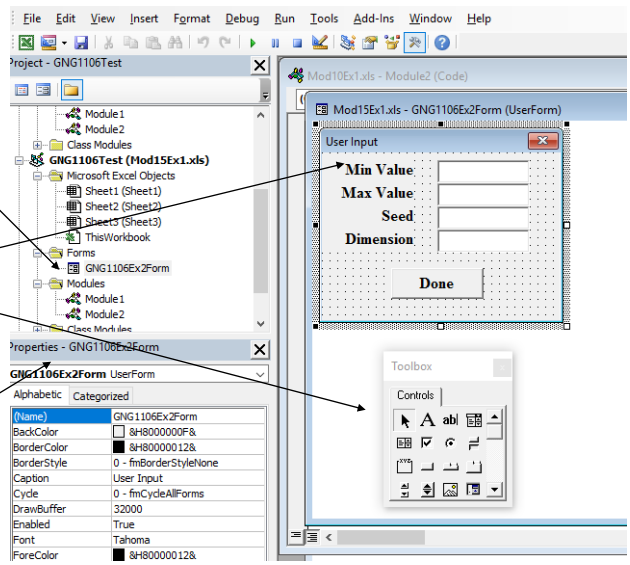


Example 2 (return to example 1)

- Problem Statement
  - To transpose a square matrix (n X n)
- Gathering of Information and Description of Input/Output
  - Engineering uses matrices, for example, they can be used to analyze the effect of earthquakes on structures (Matrix Analysis of Structural Dynamics: Applications and Earthquake Engineering, Franklin Y. Cheng, CRC Press, 2000)
  - Transposing a matrix consists of moving the values in the rows into the corresponding columns. For example, if $M = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$, then the transposed matrix is $M^T = \begin{matrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{matrix}$
  - Input: Dimension of the square matrix (maximum 10), minimum value, maximum value, and seed for generating random numbers.
  - Output: the matrix and the transposed matrix
- This time, we use a form to get the input from the user

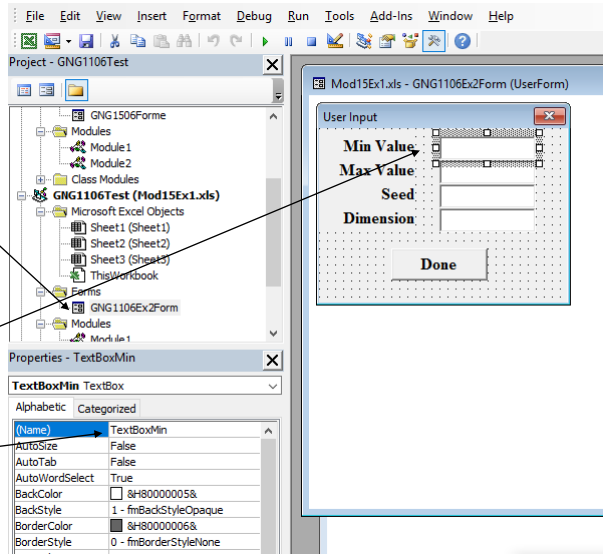
Developing a VB Form

- Project Window
  - Shows the form GNGEx2Form and Modules (contain macro code)
- Form with caption User Input
  - Note the ToolBox is available to add controls
- Properties Window
  - Not where the name of the Form is defined



Developing a VB Form

- Project Window
  - Shows the form GNGEx2Form and Modules (contain macro code)
- Form with caption User Input
  - Note that a text box is selected
- Properties Window
  - Note where the name of the text box is defined



GNG1106Ex2 – Input