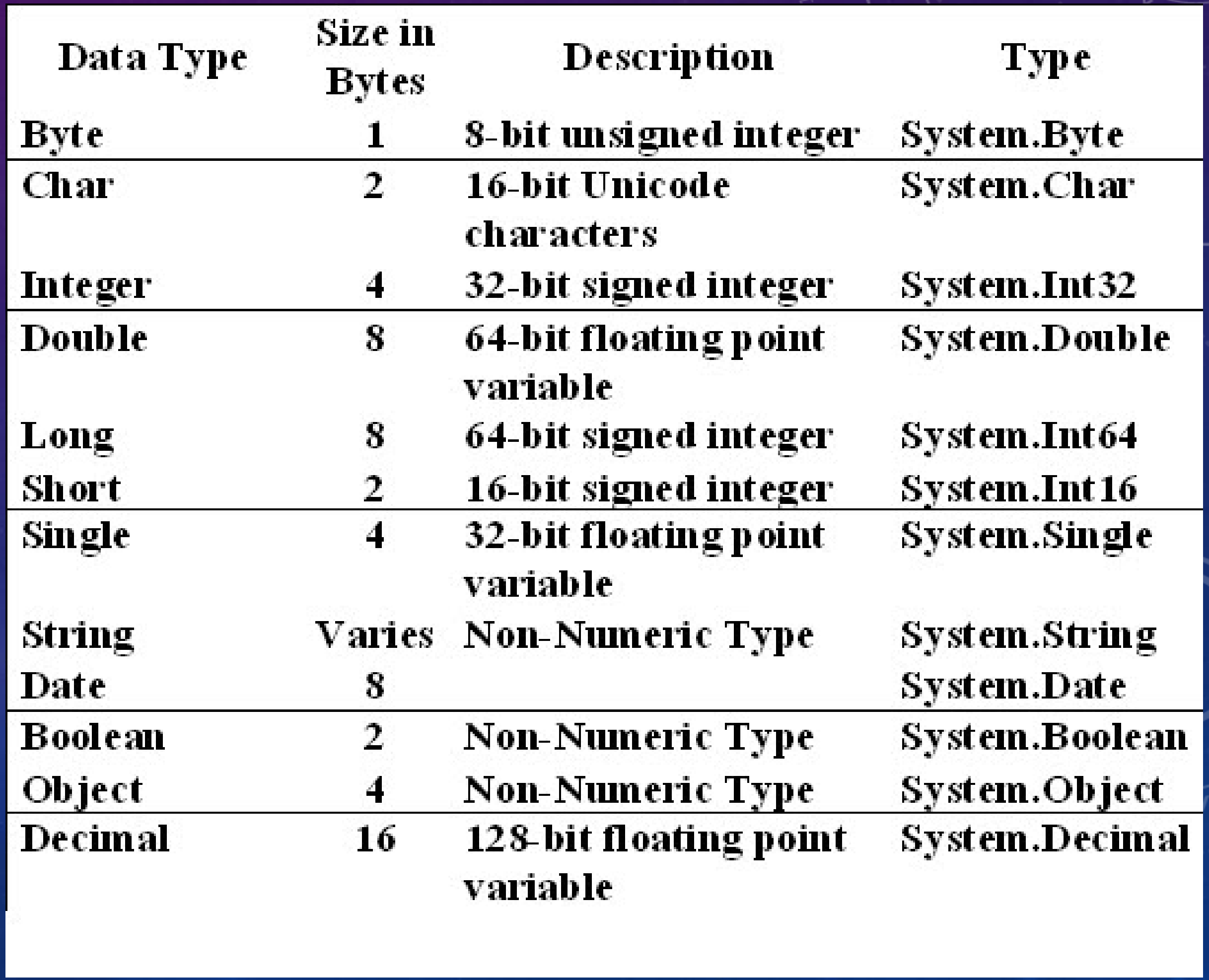
DATA TYPES & ARRAYS

VISUAL BASIC

TABLE OF CONTENTS

LECTURE CONTEXT

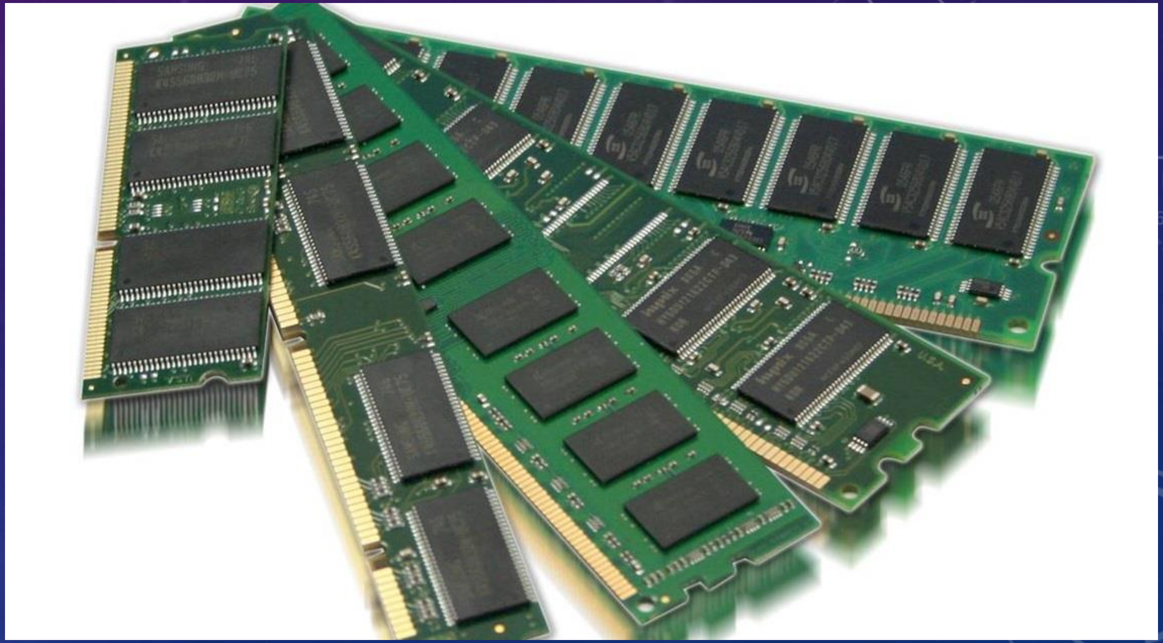
- Be familiar with the data types in VB2010 and how to use them
- Be familiar with the concept of 'scope' and how it helps programmers avoid accidently overwriting data
- Understand how to create an array of data
- Be able to iterate through an array to fill or investigate its contents



Data Type	Size in Bytes	Description	Type
Byte	1	8-bit unsigned integer	System.Byte
Char	2	16-bit Unicode characters	System.Char
Integer	4	32-bit signed integer	System.Int32
Double	8	64-bit floating point variable	System.Double
Long	8	64-bit signed integer	System.Int64
Short	2	16-bit signed integer	System.Int16
Single	4	32-bit floating point variable	System.Single
String	Varies	Non-Numeric Type	System.String
Date	8		System.Date
Boolean	2	Non-Numeric Type	System.Boolean
Object	4	Non-Numeric Type	System.Object
Decimal	16	128-bit floating point variable	System.Decimal

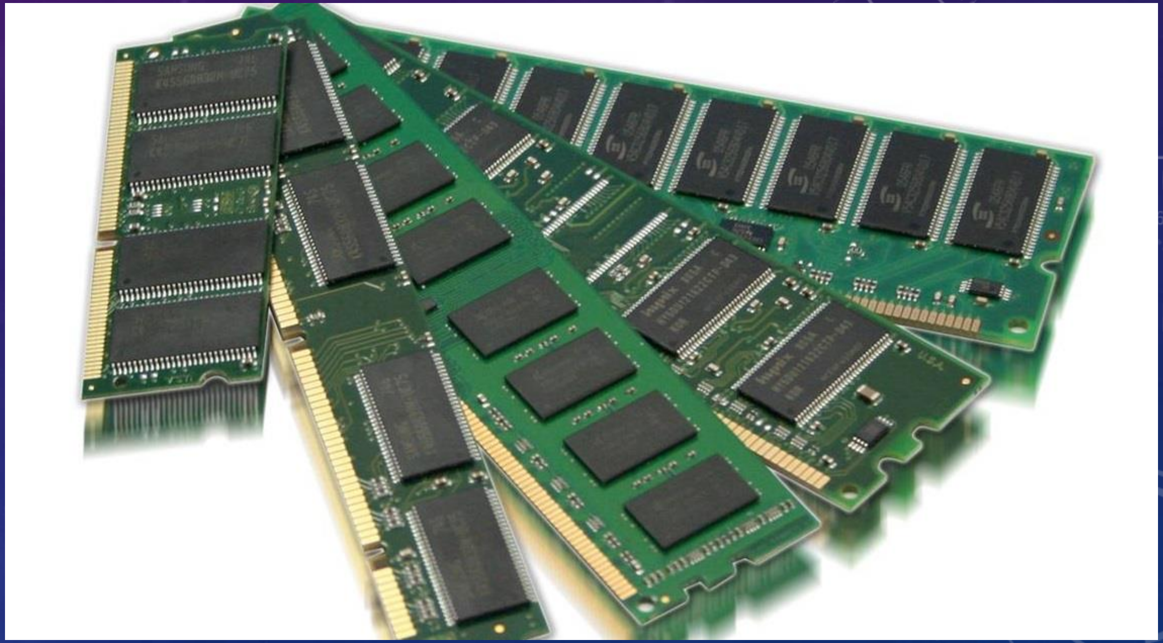
LECTURE CONTEXT

- Programs use memory (RAM) to store data and actually run (they are hosted in memory when running). Engineering applications typically use a lot of memory and store a large number of data, typically contiguous time-sampled data in instrumentation applications.
- To efficiently utilize memory and ensure no data errors occur, it is important to understand the various variable types available to the programmer.
- Also, it is useful to understand how to store ‘clumps’ or ‘groups’ of data and manipulate them as a single entity. This lecture looks at how data is stored, how to limit accidental overwrites and how to manipulate it in arrays.



WHAT VARIABLES CAN STORE

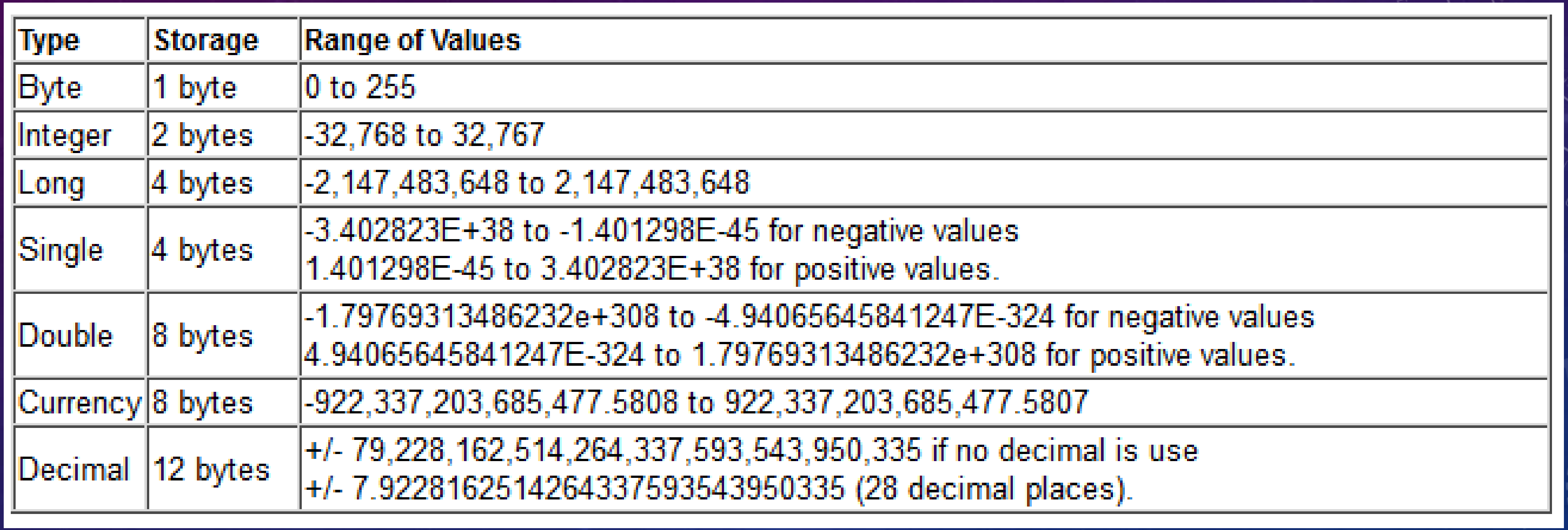
- 99.999% of programs work on data of some sort, usually both manipulating and storing it
- Part B Matlab/C# modules introduced students to variables: referred to as data ‘Types’
- Data containers of varying size and ‘shape’
- VB supports the same variable Types as all other common languages
- Numeric data types store numbers, that can be treated as numbers, i.e. +-/* SQRT and so on
- Visual Basic 2010, C++, C#, F# etc. divides these into 7 types, depending on the range of values they can store and memory they use



WHAT VARIABLES CAN STORE

- Most code uses Integer Type as the most efficient type: only hold +/- whole numbers
- Double Type used for Maths and high precision – all Maths Functions such as Sin, Tan, SQRT etc. require Doubles
- Decimal Type stores values with highest range and precision
- Text is stored in a String Type – **NB** the characters “3434” stored in a String is NOT a number, it is a string of characters = a ‘word’

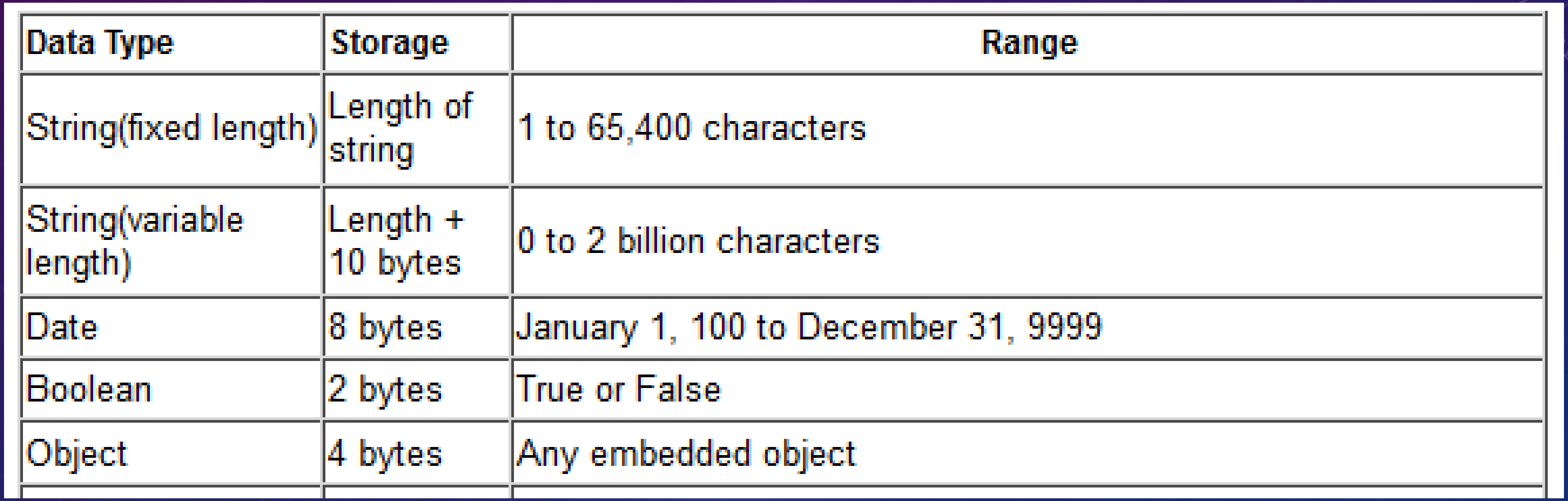
NUMERIC TYPES



Type	Storage	Range of Values
Byte	1 byte	0 to 255
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E+38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).

e.g. Dim intLoopCounter as Integer

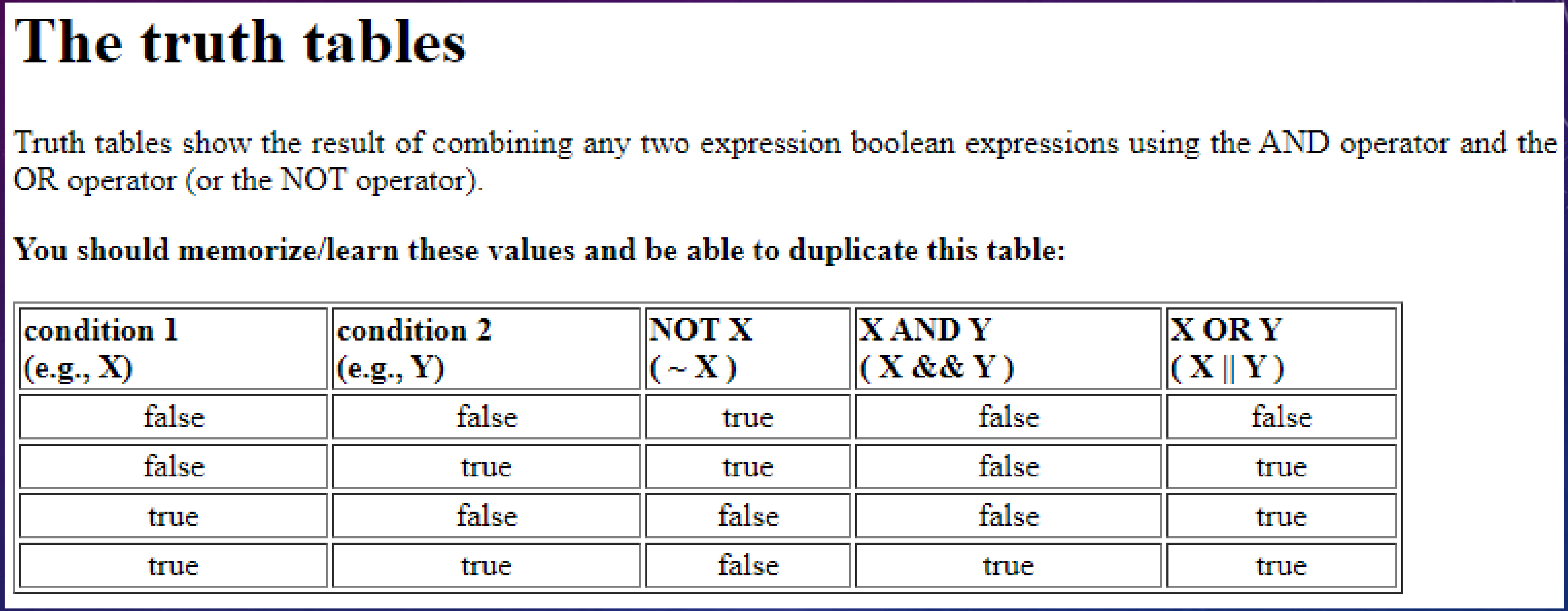
NON-NUMERIC TYPES



Data Type	Storage	Range
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999
Boolean	2 bytes	True or False
Object	4 bytes	Any embedded object

e.g. Dim myOutcome as String

TRUTH TABLES



The truth tables

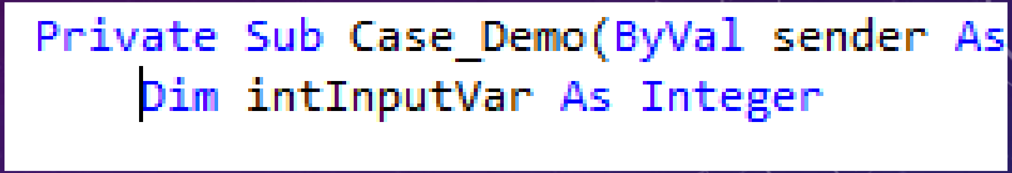
Truth tables show the result of combining any two expression boolean expressions using the AND operator and the OR operator (or the NOT operator).

You should memorize/learn these values and be able to duplicate this table:

condition 1 (e.g., X)	**condition 2 (e.g., Y)**	**NOT X (~ X)**	**X AND Y (X && Y)**	**X OR Y (X || Y)**
false	false	true	false	false
false	true	true	false	true
true	false	false	false	true
true	true	false	true	true

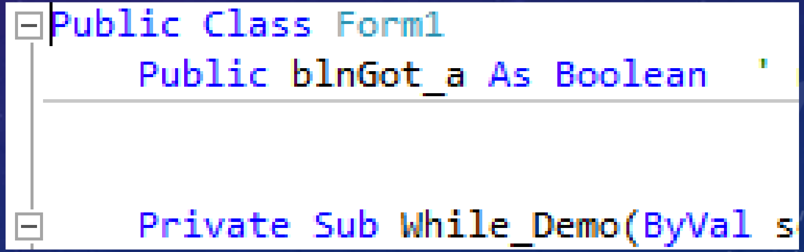
e.g. True And False should return False

VARIABLE SCOPING



```
Private Sub Case_Demo(ByVal sender As
    Dim intInputVar As Integer
```

Procedural declaration immediately after 'Private Sub' – scope is ONLY the procedure itself



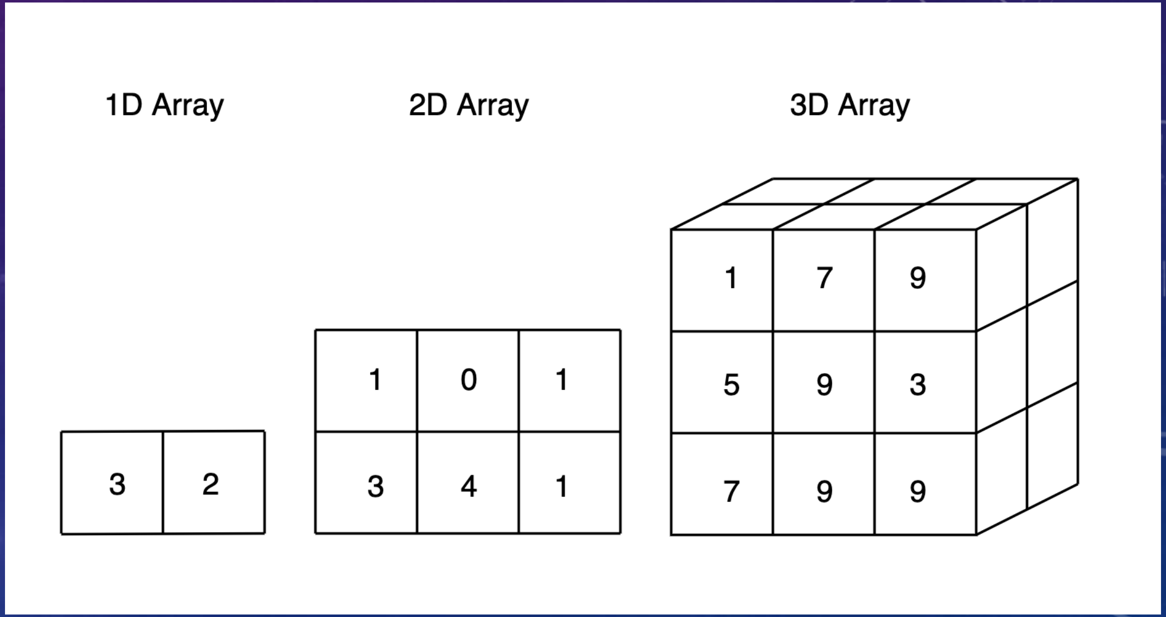
```
Public Class Form1
    Public blnGot_a As Boolean   '


    Private Sub While_Demo(ByVal s
```

Global declaration immediately after 'Public Class' – scope is ALL procedures on Form Class

- Scope of Procedural declaration is the Procedure – i.e. Private Sub to End Sub
- Local variables created at the **Dim** line and then destroyed at **End Sub** – re-use resources
- Scope of Global declaration is all code across the whole **Form** – created when program first runs and destroyed when program ends

GROUPING VARIABLES

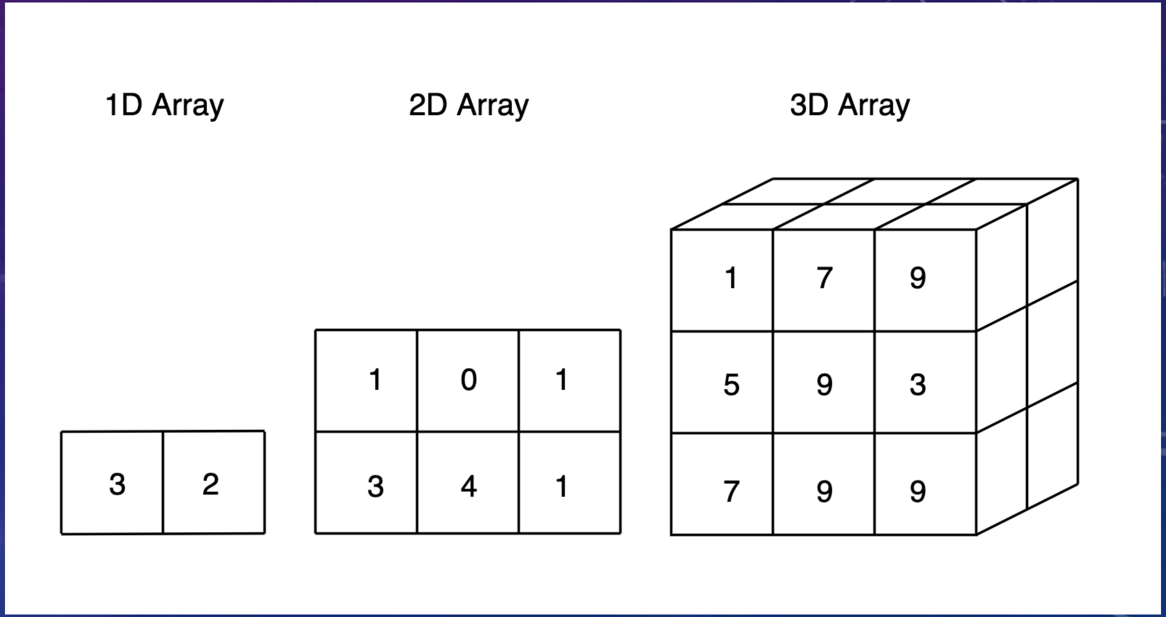
- Large programs might require 1000+ variables – easily introduce errors and difficult to track!
- Engineering applications in particular often deal with 1000s of measurement or model data
- Difficult to keep track of, and *name* that many individual variables and errors may appear
- Manage them by grouping and/or limiting *their* ***Scope*** across the application
- In visual basic and a lot of other programming languages it is treated a group of Types



GROUPING VARIABLES

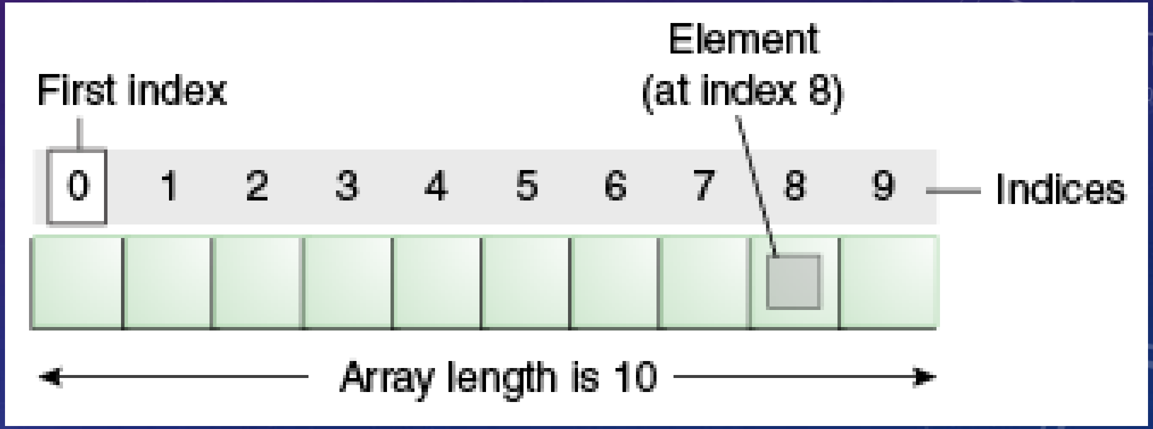
- An Array is a group of variables that contains a set of **data items of same type** and name
- Arrays have a Name, Type, Size and Dimension

  e.g. Dim *intData(n) as Integer*
- Stored data is accessed by ***index (n)*** where n is *the index of the last cell* ***(n+1 cells)***
- Range = from LowerBound to UpperBound where UpperBound is index of last cell (n)
- We can use single or multi dimensional: 2D arrays are common for measurement data and images and have Rows and Columns



ARRAY INDEXING

- May be declared as **dynamic** with no initial defined range allowing array to size to data
- Size changes by re-dimensioning at each use to fit data: typically using an incremented variable
  E.g. Dim intDataArray() as integer
- *Before use, re-dimension*
  ReDim intDataArray(intArraySize)
- *intArraySize = intArraySize + 1*
  ‘increase index pointer ready for next loop
- *ReDim sets array to zero – use Preserve keyword to retain data on ReDim*

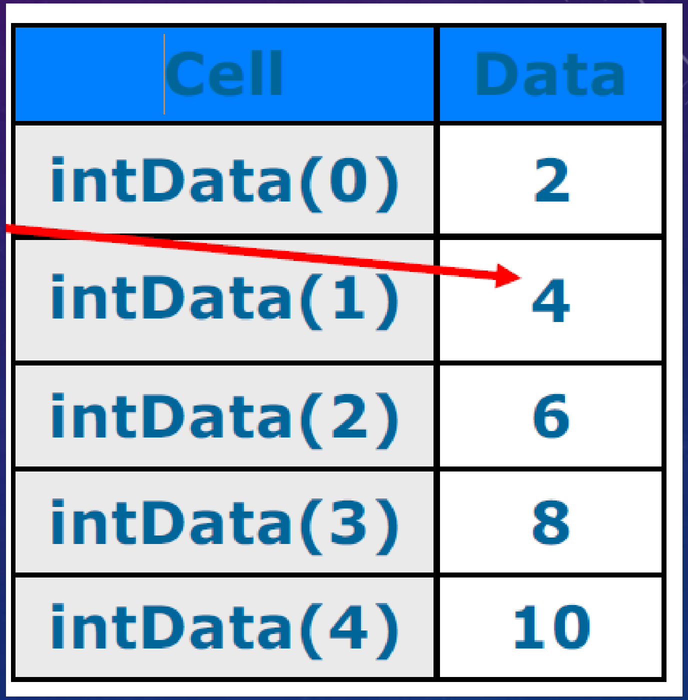


ARRAY VISUALIZATION

- A 1 dimensional array may be visualized as a single spreadsheet column
  E.g. Dim intData(4) as integer
- Contains 5 (4+1) contiguous integer elements
- May fill cells with data thus:
  intData(0)=2, intData(1)=4,

  intData(2)=6, intData(3)=8,

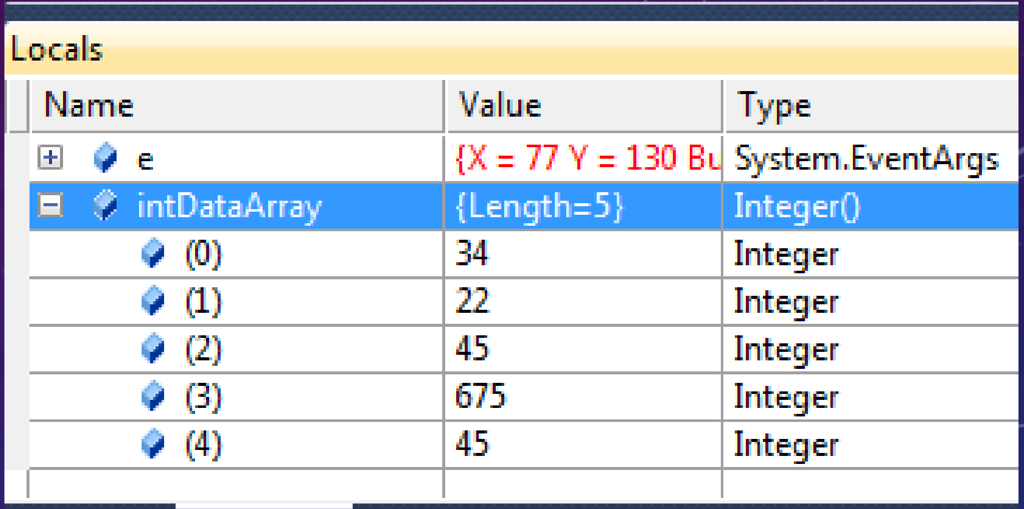
  intData(4)=10
- Dim intData as Integer = {2,4,6,8,10}



Cell	Data
intData(0)	2
intData(1)	4
intData(2)	6
intData(3)	8
intData(4)	10

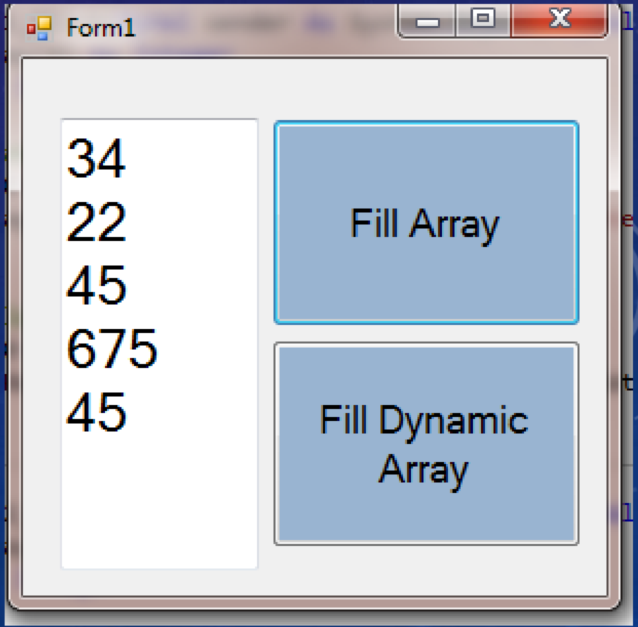
CODE TO FILL ARRAY WITH DATA

Usual method is iteration *(mostly For loops are generally used when one knows the size)*



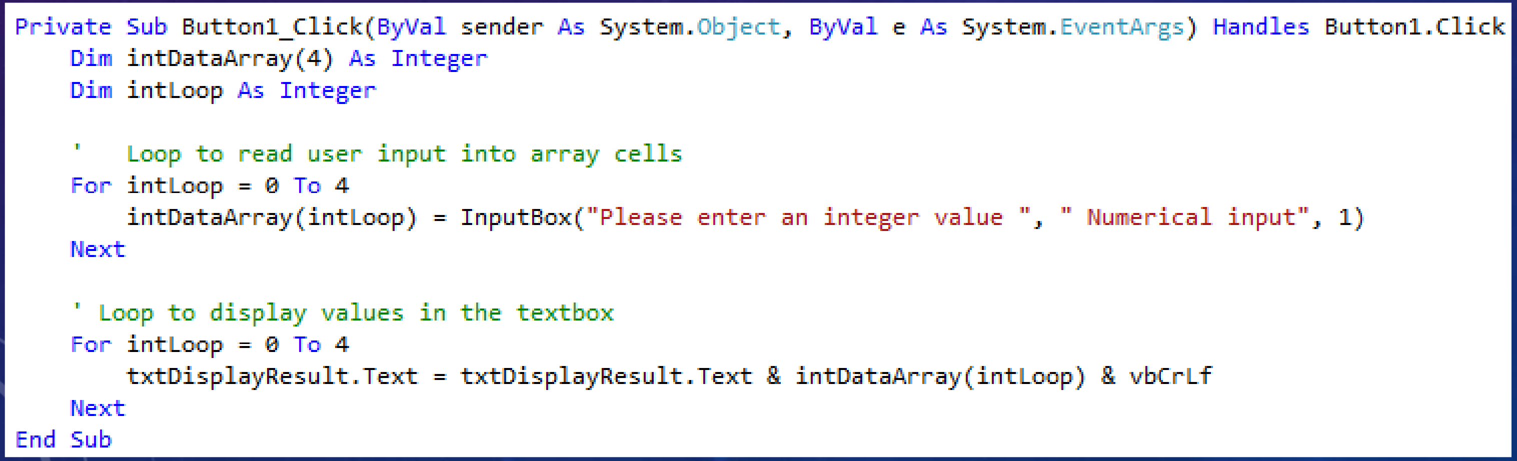
Locals

Name	Value	Type
e	{X = 77 Y = 130 Bu	System.EventArgs
intDataArray	{Length=5}	Integer()
(0)	34	Integer
(1)	22	Integer
(2)	45	Integer
(3)	675	Integer
(4)	45	Integer



Form1

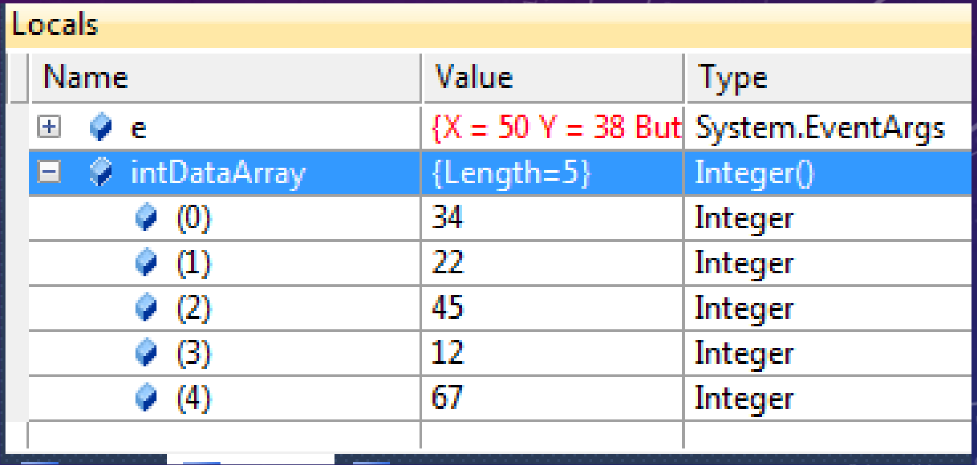
```
34
22
45
675
45
```



```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim intDataArray(4) As Integer
    Dim intLoop As Integer

    '   Loop to read user input into array cells
    For intLoop = 0 To 4
        intDataArray(intLoop) = InputBox("Please enter an integer value ", " Numerical input", 1)
    Next

    ' Loop to display values in the textbox
    For intLoop = 0 To 4
        txtDisplayResult.Text = txtDisplayResult.Text & intDataArray(intLoop) & vbCrLf
    Next
End Sub
```

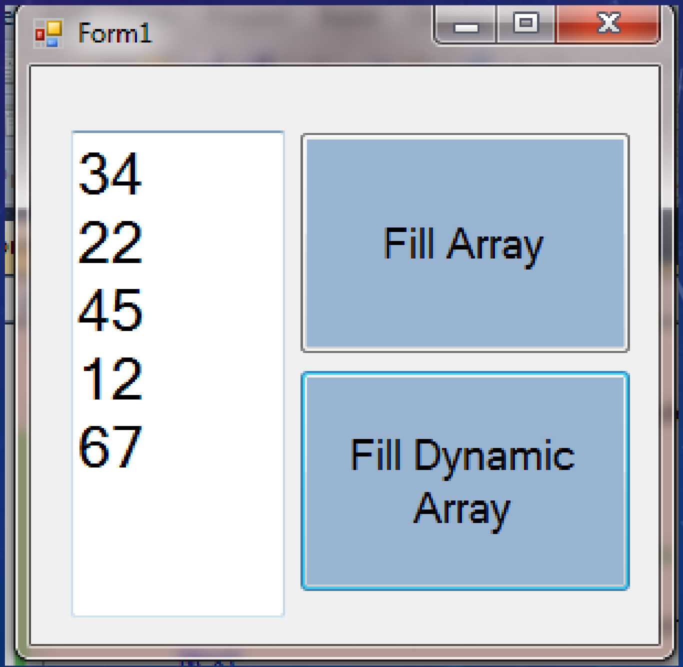
CODE TO FILL A DYNAMIC ARRAY

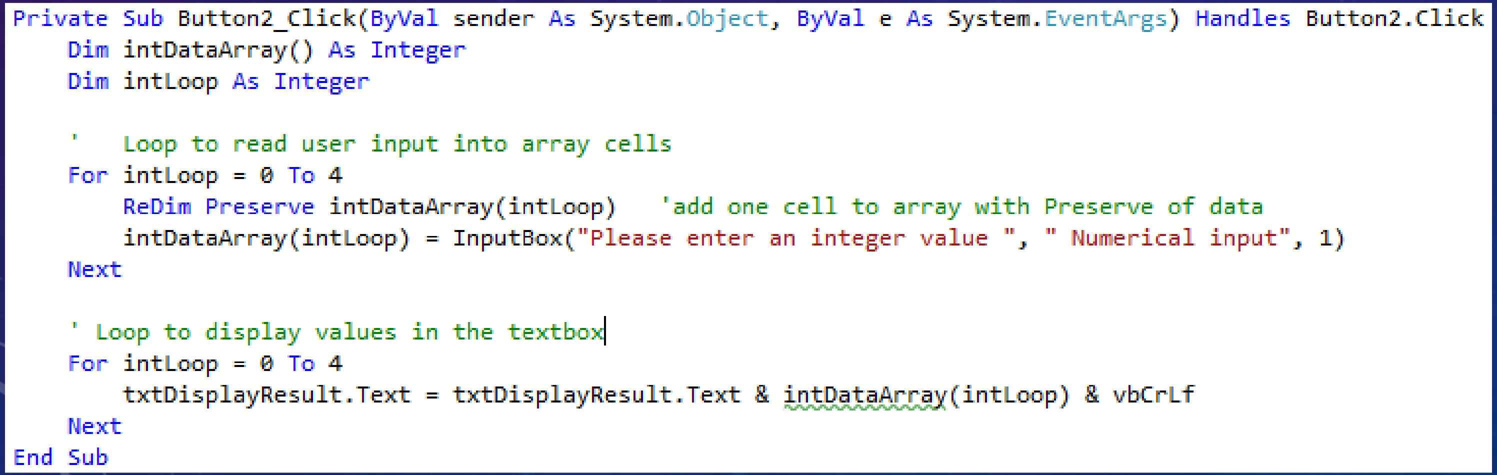
- Note use of Preserve Keyword to prevent all cells being set to zero during array ReDim



Locals

Name	Value	Type
e	{X = 50 Y = 38 But	System.EventArgs
intDataArray	{Length=5}	Integer()
(0)	34	Integer
(1)	22	Integer
(2)	45	Integer
(3)	12	Integer
(4)	67	Integer

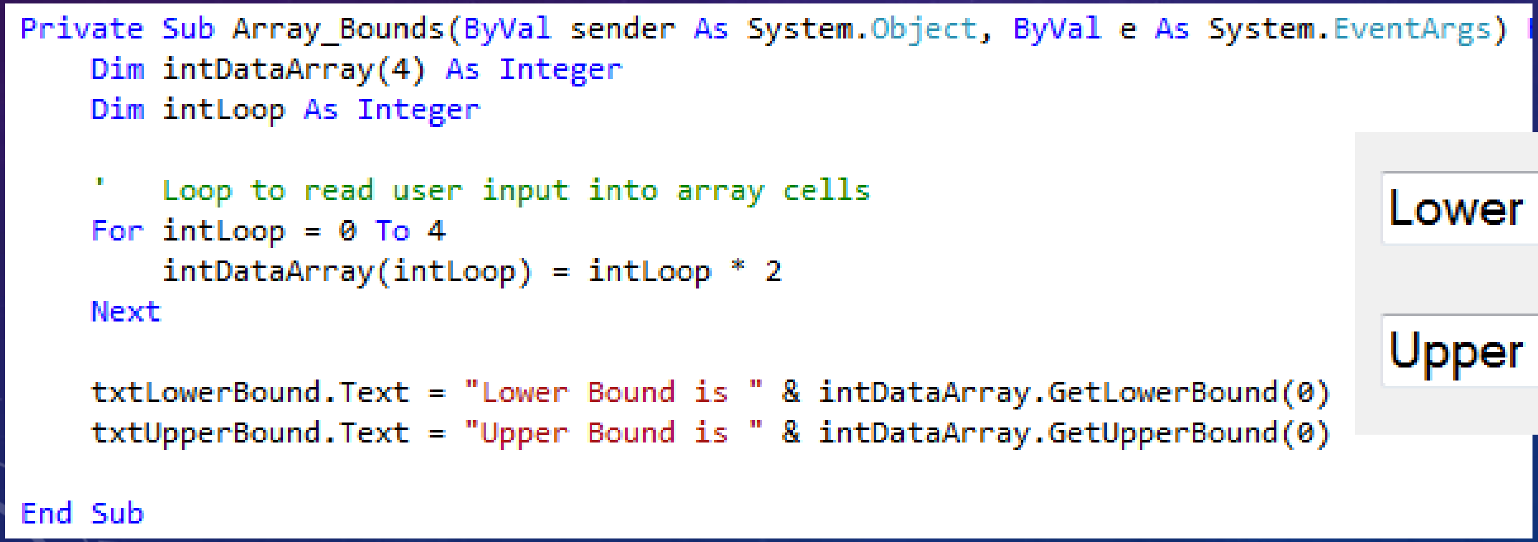




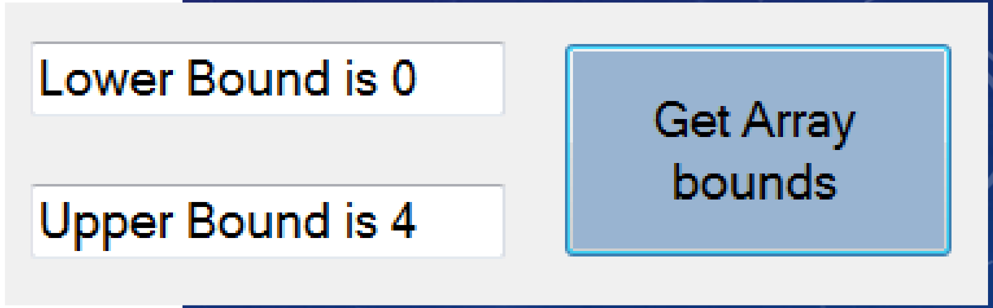
```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    Dim intDataArray() As Integer
    Dim intLoop As Integer

    '   Loop to read user input into array cells
    For intLoop = 0 To 4
        ReDim Preserve intDataArray(intLoop)   'add one cell to array with Preserve of data
        intDataArray(intLoop) = InputBox("Please enter an integer value ", " Numerical input", 1)
    Next

    ' Loop to display values in the textbox
    For intLoop = 0 To 4
        txtDisplayResult.Text = txtDisplayResult.Text & intDataArray(intLoop) & vbCrLf
    Next
End Sub
```

WORKING WITH BOUNDS

- One may work with Upper or Lower Bounds rather than known values: good for dynamic arrays when size is not always known
- Use GetUpperBound() or GetLowerBound() Methods to find array size
- Use 0 as argument for 1D and 0 or 1 for 2D



```
Private Sub Array_Bounds(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Dim intDataArray(4) As Integer
    Dim intLoop As Integer

    '   Loop to read user input into array cells
    For intLoop = 0 To 4
        intDataArray(intLoop) = intLoop * 2
    Next

    txtLowerBound.Text = "Lower Bound is " & intDataArray.GetLowerBound(0)
    txtUpperBound.Text = "Upper Bound is " & intDataArray.GetUpperBound(0)

End Sub
```

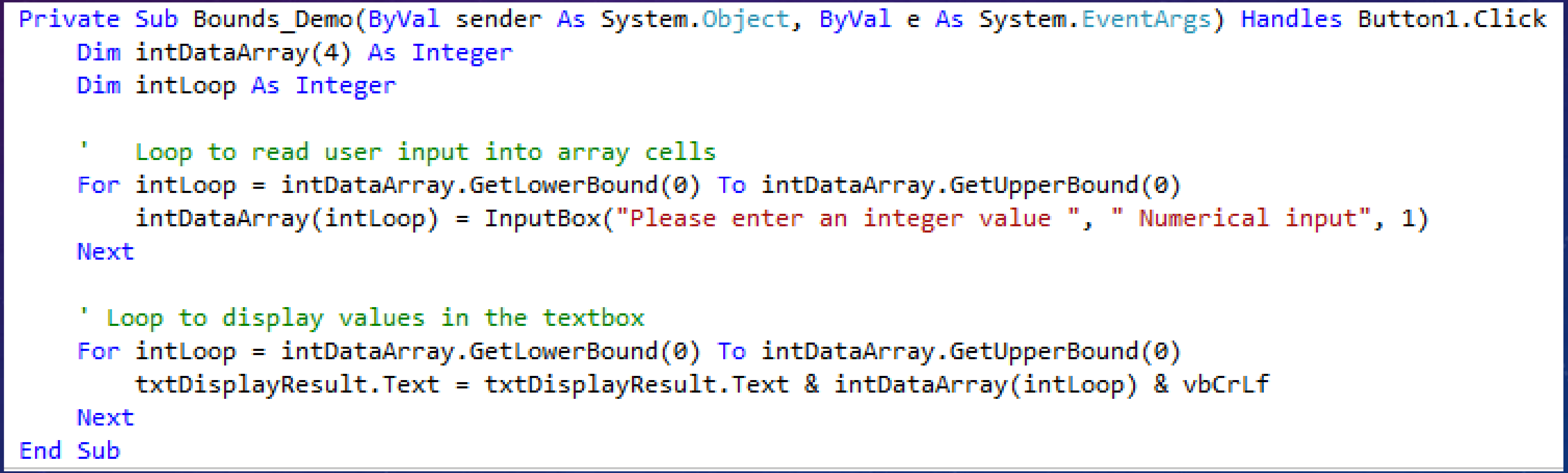


Lower Bound is 0

Upper Bound is 4

WORKING WITH BOUNDS

- Using Upper and Lower Bounds makes code changes easier to implement
- When one changes the size of a static array declaration, one doesn't then have to change all loop limits within the actual code



```
Private Sub Bounds_Demo(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim intDataArray(4) As Integer
    Dim intLoop As Integer

    '   Loop to read user input into array cells
    For intLoop = intDataArray.GetLowerBound(0) To intDataArray.GetUpperBound(0)
        intDataArray(intLoop) = InputBox("Please enter an integer value ", " Numerical input", 1)
    Next

    ' Loop to display values in the textbox
    For intLoop = intDataArray.GetLowerBound(0) To intDataArray.GetUpperBound(0)
        txtDisplayResult.Text = txtDisplayResult.Text & intDataArray(intLoop) & vbCrLf
    Next
End Sub
```

HANDS ON EXERCISE

- We will practice our truth tables
- We will use this concept a lot in our work

No	Operations	Outcome
1	True or True	
2	True or False	
3	True And False	
4	True And True	
5	True or False or True	
6	True And True And False	