# Language Fundamentals

## VBA Concepts

Sept. 2013

CEE 3804 Faculty

## Language Fundamentals

1. Statements
2. Data Types
3. Variables and Constants
4. Functions
5. Subroutines

## Data Types

1. Numeric
   - Integer
   - Long integer
   - Single-precision
   - Double-precision
   - Currency
2. Character Strings
   - Variable length
   - Fixed length
3. Variant in VBA

## Numeric Data Types, VBA

- **Integers in VBA, Short in VB***
  - **Two bytes**
  - **32,767 to -32,768**
- **Long Integers in VBA, Integers in VB***
  - **Four bytes**
  - **2,147,483,647 to -2,147,483,648**
- **Single Precision**
  - **Four bytes**
  - **-3.402833E+38 to -1.40129E-45 negative**
  - **1.40129E-45 to 3.402823E+32  positive**
- **Double Precision**
  - **Eight bytes**
  - **-1.7977E+308 to -4.9407E-324 negative**
  - **4.9407E-324 to 1.7977E+308 positive**

* Integer types in VB are larger than same named types in VBA

## Integers

- Integers are whole numbers
- They may be either positive or negative
- The range of VBA integers is -32767 to +32787
- If the number if positive it is not necessary to put a + sign in front of the number
- You cannot use commas to separate groups of digits in an integer
- Examples of valid integers are:

```
    1234         -13550        1667        -340
     15          -30000       -5002         221
```

- Examples of invalid integers are:

```
-50000 35000 3,456 3.123
```

## Long Integers

- A long integer can hold a whole number but it can store a larger range of values
- VBA uses “Long” for this data type, VB uses “Integer”
- The range of values that can be stored in a long integer is -2,147,483,648 to +2,147,483,648
- To declare a long integer variable you use the following declaration statement

```
DIM x as Long
```

- Examples of some long integer constants are the following

```
32895          65483        -68957
56658        -1236978        58960321
```

## Single Precision

- A real number is a number which contains a decimal point. In Basic real numbers can be represented as single precision numbers or as double precision numbers
- Real numbers cannot be represented in a computer exactly - only approximately
- Single precision numbers are accurate to the first seven digits
- 4 bytes of storage space

## Single Precision Numbers

- Single precision numbers can be written using fixed point notation or exponential  notation
- The range of single precision numbers is from approximately -3.4E+38 to +3.4E+38
- Examples of single precision numbers are:

```
102.3          0.00034        5678.90
10.5e-3        100E10         -5.67e-35
22!              -35!          12.4e-27
```

## Double Precision Numbers

- Double precision numbers are accurate to 15 or 16 digits
- 8 bytes of storage space
- The range of double precision numbers is from approximately -1.8D+308 to +1.8D-308
- The letter D is used to represent the exponent (instead of the letter E)
- Examples of double precision numbers are:

```
12345.6789012345     0.000012345036953
10.24D-300           12.687D+300
```

## Currency

- This data type is designed specifically to store financial information
- Currency values are in fixed-point format with up to 15 digits before the decimal and 4 digits after the decimal point
- The range of numbers is from 922,337,203,685,477.5805, to +922,337,203,685,477.5807

## Strings

- There are two types of stings:
  - variable length
  - fixed length
- In a variable length string, the stored string can be of any length (up to 65,500 characters)
- Sequence of alphanumeric characters enclosed by double quotation marks

```
 "What is your name?"
"Enter yield strength of steel"
```

## Strings

- A fixed length string stores a string value of a predetermined length.
- You declare the fixed length to be from 0 to 32,767 characters.
- The length of the string cannot change

## Variant Data Type

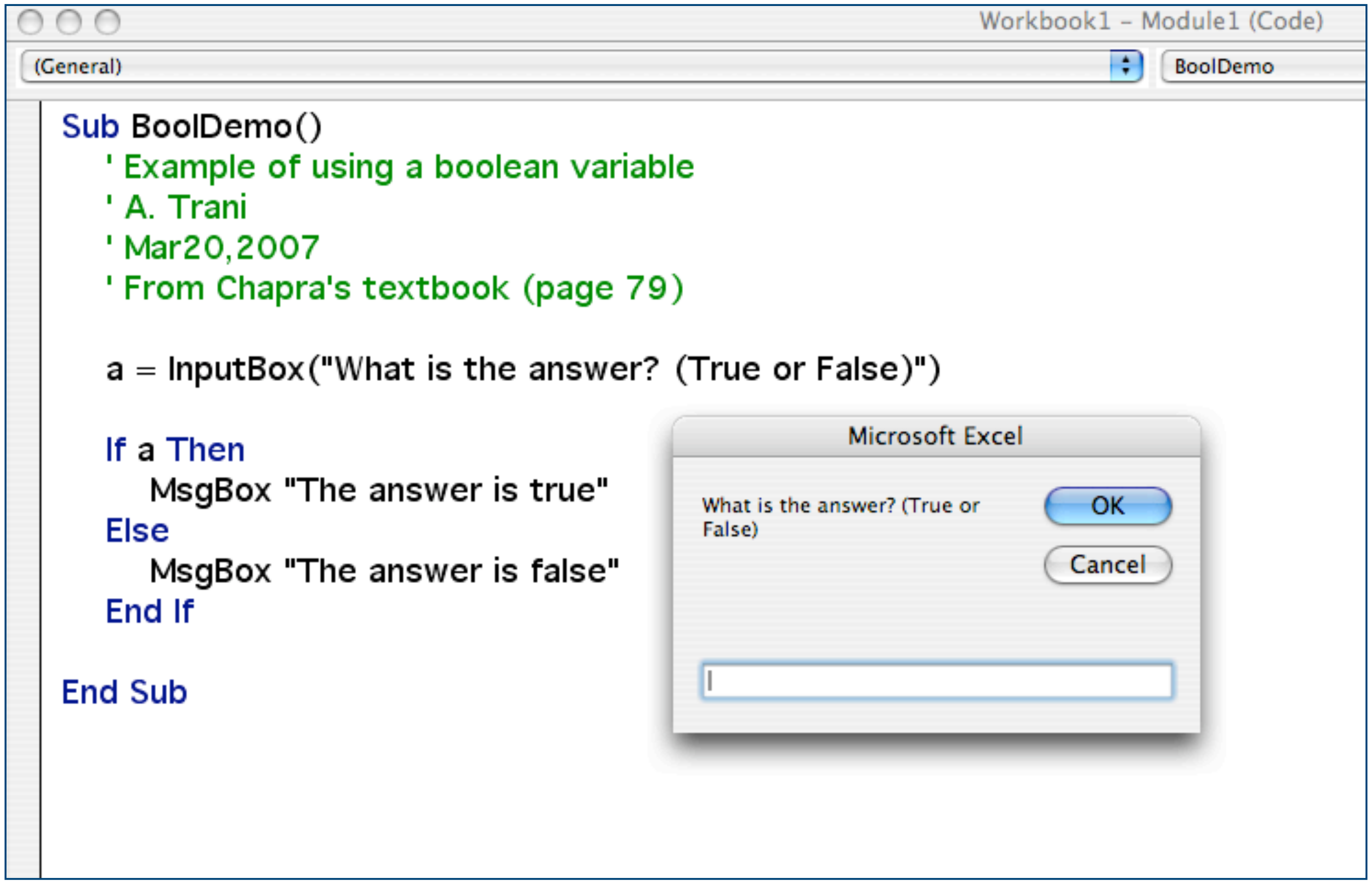
- A variant is a data type that can store data of any other type
- The stored value can be numeric, string, or date/time information
- Visual Basic automatically converts a value stored in a Variant variable to the necessary data type

```
Dim x as Variant
x = “35”     ‘ string stored
Print 2 * x ‘ string converted
```

- A variant variable actually stores two pieces of information: a value, and a code  number indicating its data type

## Boolean Data Type

- Logical constants that take two possible values:
  - True
  - False
- Example

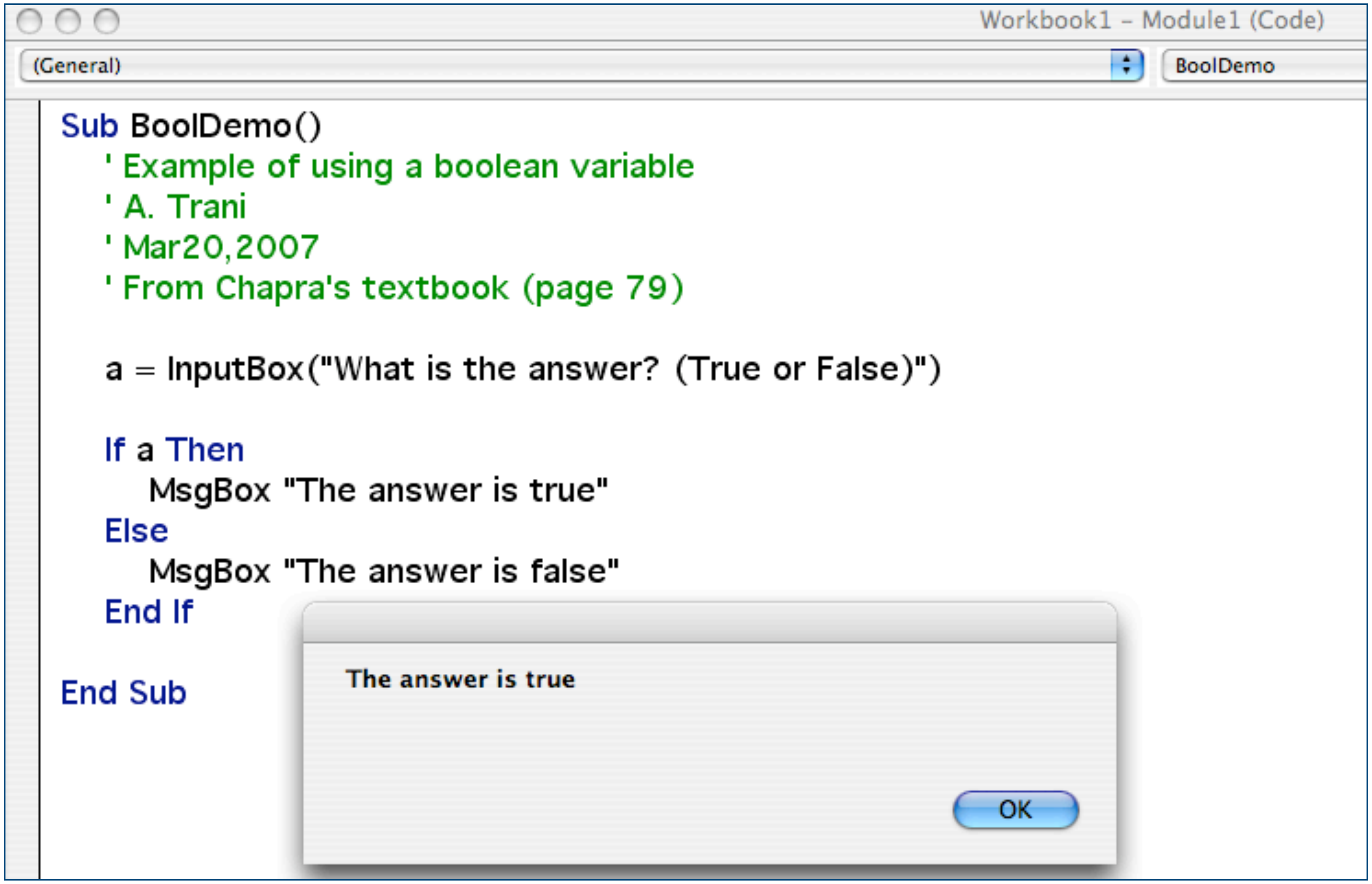


```
Sub BoolDemo()
    ' Example of using a boolean variable
    ' A. Trani
    ' Mar20,2007
    ' From Chapra's textbook (page 79)

    a = InputBox("What is the answer?  (True or False)")

    If a Then
        MsgBox "The answer is true"
    Else
        MsgBox "The answer is false"
    End If

End Sub
```

Microsoft Excel

What is the answer? (True or False)

## Boolean Example



```
Sub BoolDemo()
    ' Example of using a boolean variable
    ' A. Trani
    ' Mar20,2007
    ' From Chapra's textbook (page 79)

    a = InputBox("What is the answer?  (True or False)")

    If a Then
        MsgBox "The answer is true"
    Else
        MsgBox "The answer is false"
    End If

End Sub
```

```
The answer is true
```

## Variables

- Permit us to access data stored in memory through the use of symbolic names
- Value of a variable can change during program execution
- Made up of the letters A to Z, digits 0 to 9, and underscore (_).
- First character must be a letter of the alphabet.  Cannot begin with a number or a period
- Can be from 1 to 40 characters long
- Cannot match any reserved words such as
  - `If        Then      Input       While`
- May end with the symbols %, !, #, $, & and @

## Variable Names

### Suggestions for Naming Variables

- Use lower case character for first letter
- Use lowercase for remaining characters
- Uppercase for first letter of compound name

```
number      firstRoot    volume
sum         mean         surfaceArea
maxValue    minValue     counter
```

- BASIC is **NOT** type sensitive (be careful)

  `Number` `NUMBER` `NumBer` `number`

  are all considered to be the same
- Avoid short non-descriptive names

## Declaring Variables

```
Dim x As Single
Dim y As Integer
Dim z1 AS String
Dim z2 AS String*10
Dim a AS Double
Dim z as Variant
Dim bigNum as Long
Dim dollars as Currency
```

**NOTE: Type declaration characters take precedence**

## Forcing Declarations

- You can force all variables to be declared by using the following statement:

```
    Option Explicit
```

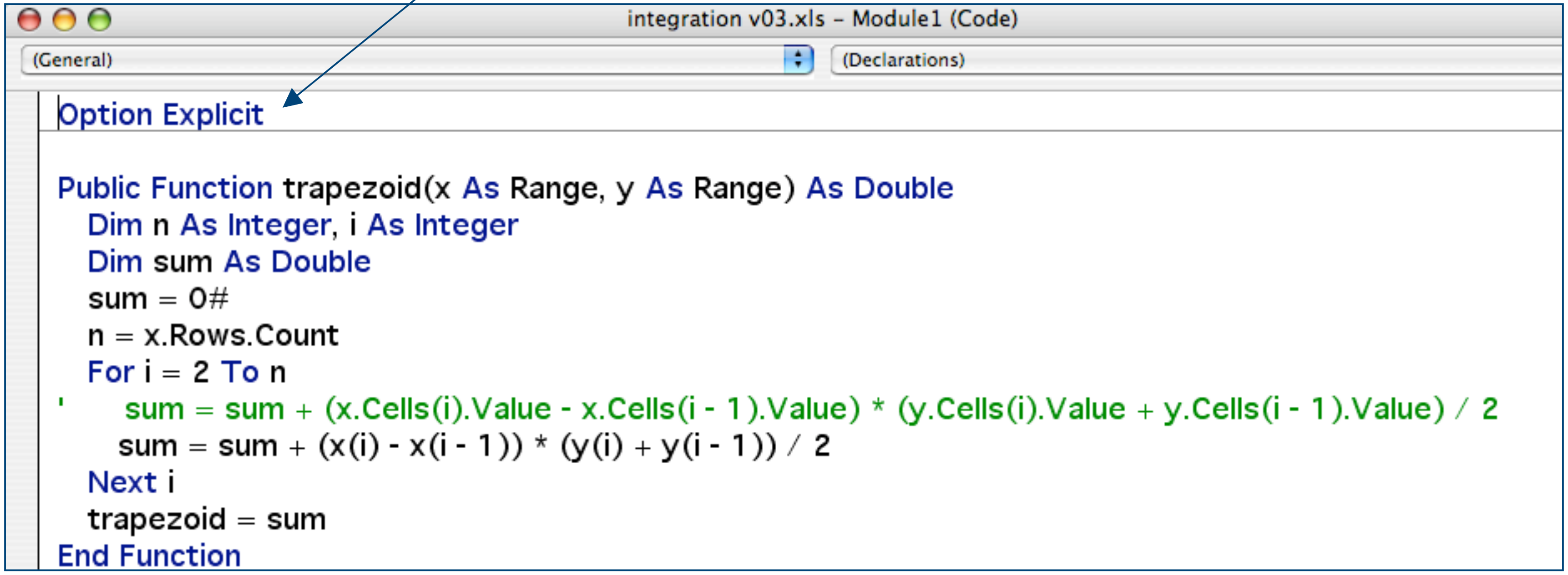
- The Option Explicit statement is placed in the Declarations section.
- It requires that all variables be declared within that form or module

## Option Explicit

- If you try to use an undeclared variable, you will get a “`Variable not declared`” error message
- You can have Visual Basic automatically place this statement in your program by specifying it in the Environment menu
- Decreases possible program "bugs"

## Example of Option Explicit

Note: all variables have to be defined in the function Trapezoid because the "Option Explicit" setting



```
Option Explicit

Public Function trapezoid(x As Range, y As Range) As Double
   Dim n As Integer, i As Integer
   Dim sum As Double
   sum = 0#
   n = x.Rows.Count
   For i = 2 To n
'     sum = sum + (x.Cells(i).Value - x.Cells(i - 1).Value) * (y.Cells(i).Value + y.Cells(i - 1).Value) / 2
      sum = sum + (x(i) - x(i - 1)) * (y(i) + y(i - 1)) / 2
   Next i
   trapezoid = sum
End Function
```

## Type Declaration Characters

**Special character at the end of a variable name which indicates the type of variable**

| Data Type | Type Declaration Character |
|---|---|
| `Integer` | `%` |
| `Long` | `&` |
| `Single-precision` | `!` |
| `Double-precision` | `#` |
| `Currency` | `@` |
| `String` | `$` |

## Type Declaration Characters

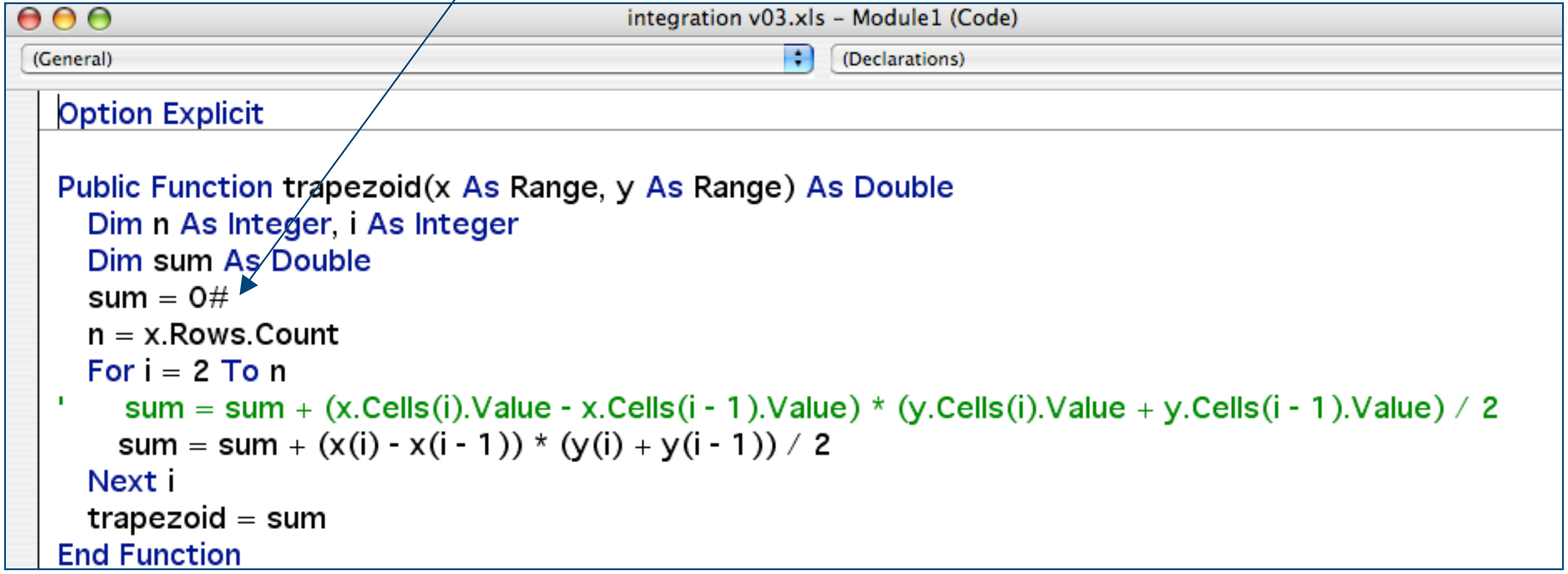
- Examples

```
count%        total!       Root!
sum#          ssNum$       yourName$
errorFlag%    longVal&     bigNum&
```

**car$, car%, car#, car&, car! are all different variables in Visual Basic for Applications**

## Example of Type Declaration

Note: variable “sum” has an explicit type declaration character. In this case sum is defined as 0.0 and VBA types 0#



```
Option Explicit

Public Function trapezoid(x As Range, y As Range) As Double
   Dim n As Integer, i As Integer
   Dim sum As Double
   sum = 0#
   n = x.Rows.Count
   For i = 2 To n
'     sum = sum + (x.Cells(i).Value - x.Cells(i - 1).Value) * (y.Cells(i).Value + y.Cells(i - 1).Value) / 2
      sum = sum + (x(i) - x(i - 1)) * (y(i) + y(i - 1)) / 2
   Next i
   trapezoid = sum
End Function
```

## Symbolic Constants

Can be used in place of numeric or string values

```
Const MAXSIZE = 50
 Dim xArray(MAXSIZE)

 Const BLACK = 0, BLUE = 1, WHITE = 15

 Const NUMCOLS = 5
```

Try using all uppercase letters for symbolic constants

In VB (not VBA) also declare data type as part of constant declaration

```
Const MAXSIZE as integer=50
Const PI as double = 3.1417
```

## VarType Function

- Use the VarType function to inspect the variable type in VBA

- Use:

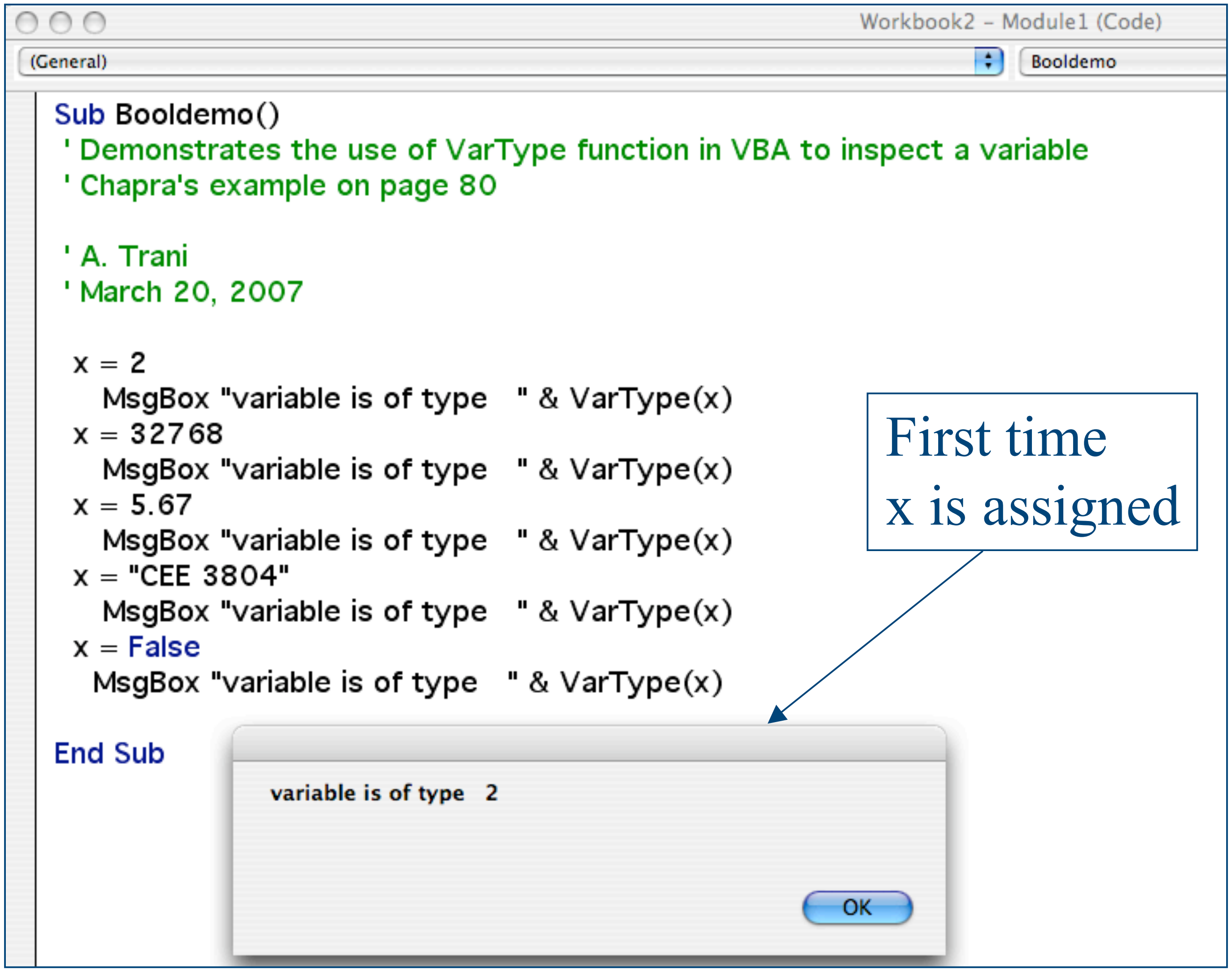
VarType (*varname*)

Where: *varname* is the variable in question

## VarType Function Codes

| Value | Variable Type |
|---|---|
| 2 | Integer |
| 3 | Long |
| 4 | Single |
| 5 | Double |
| 8 | String |
| 11 | Boolean |

## VarType Example



```
Sub Booldemo()
 ' Demonstrates the use of VarType function in VBA to inspect a variable
 ' Chapra's example on page 80

 ' A. Trani
 ' March 20, 2007

  x = 2
     MsgBox "variable is of type   " & VarType(x)
  x = 32768
     MsgBox "variable is of type   " & VarType(x)
  x = 5.67
     MsgBox "variable is of type   " & VarType(x)
  x = "CEE 3804"
     MsgBox "variable is of type   " & VarType(x)
  x = False
    MsgBox "variable is of type   " & VarType(x)

End Sub
```

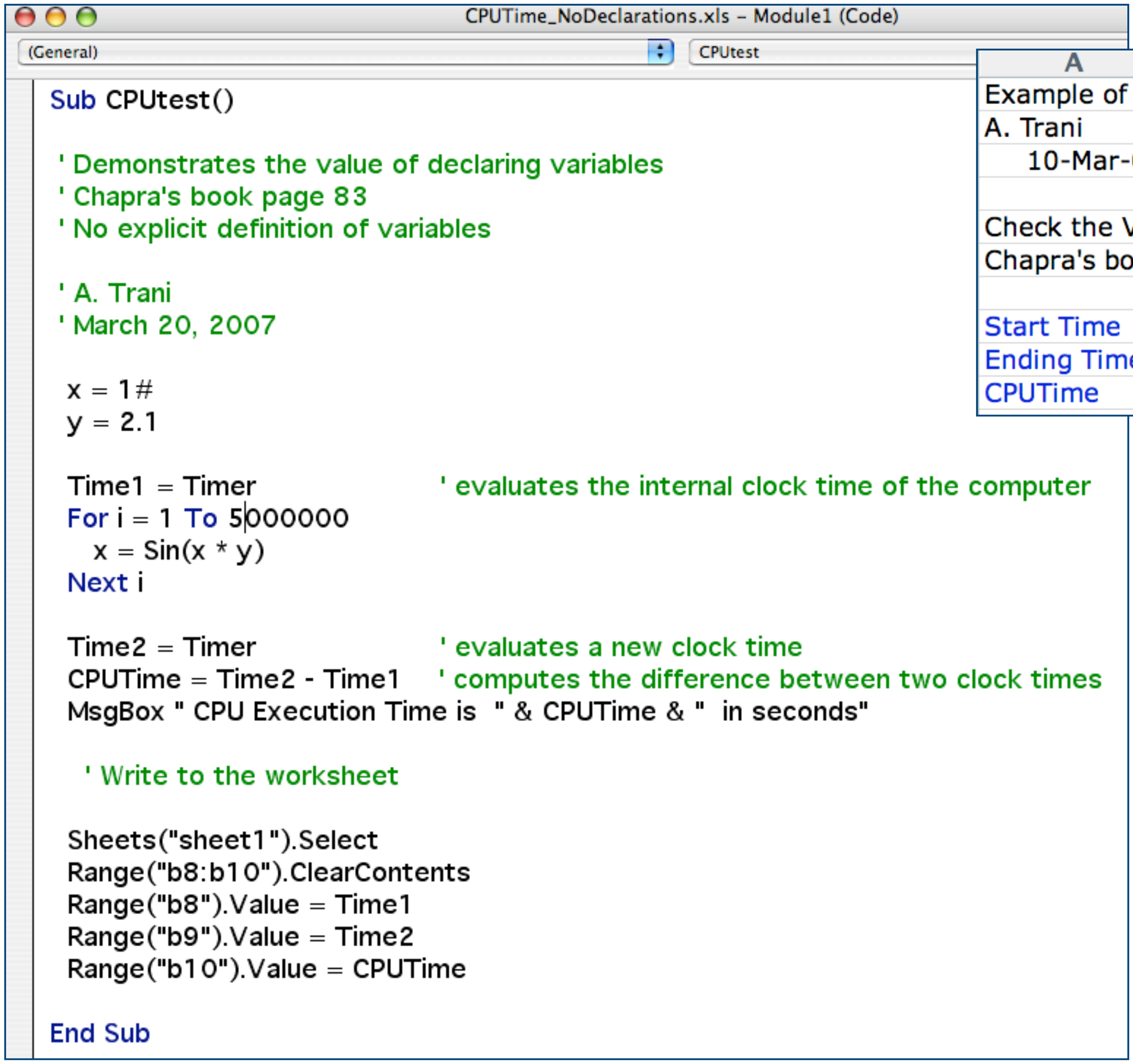
```
variable is of type   2
```

First time
x is assigned

## Computational Issues

- VBA code executes faster if variable declarations are made explicitly
- Most computers today are optimized to do floating point computations so “double” variables do not add a great deal of CPU time in execution
- In fact, declaring most variables as “double” might produce faster executions times than “single” variables

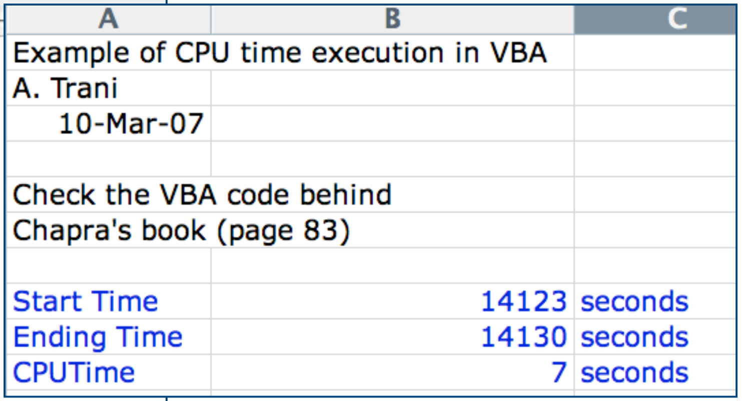
## Example (No Variable Declarations)



CPUTime_NoDeclarations.xls – Module1 (Code)

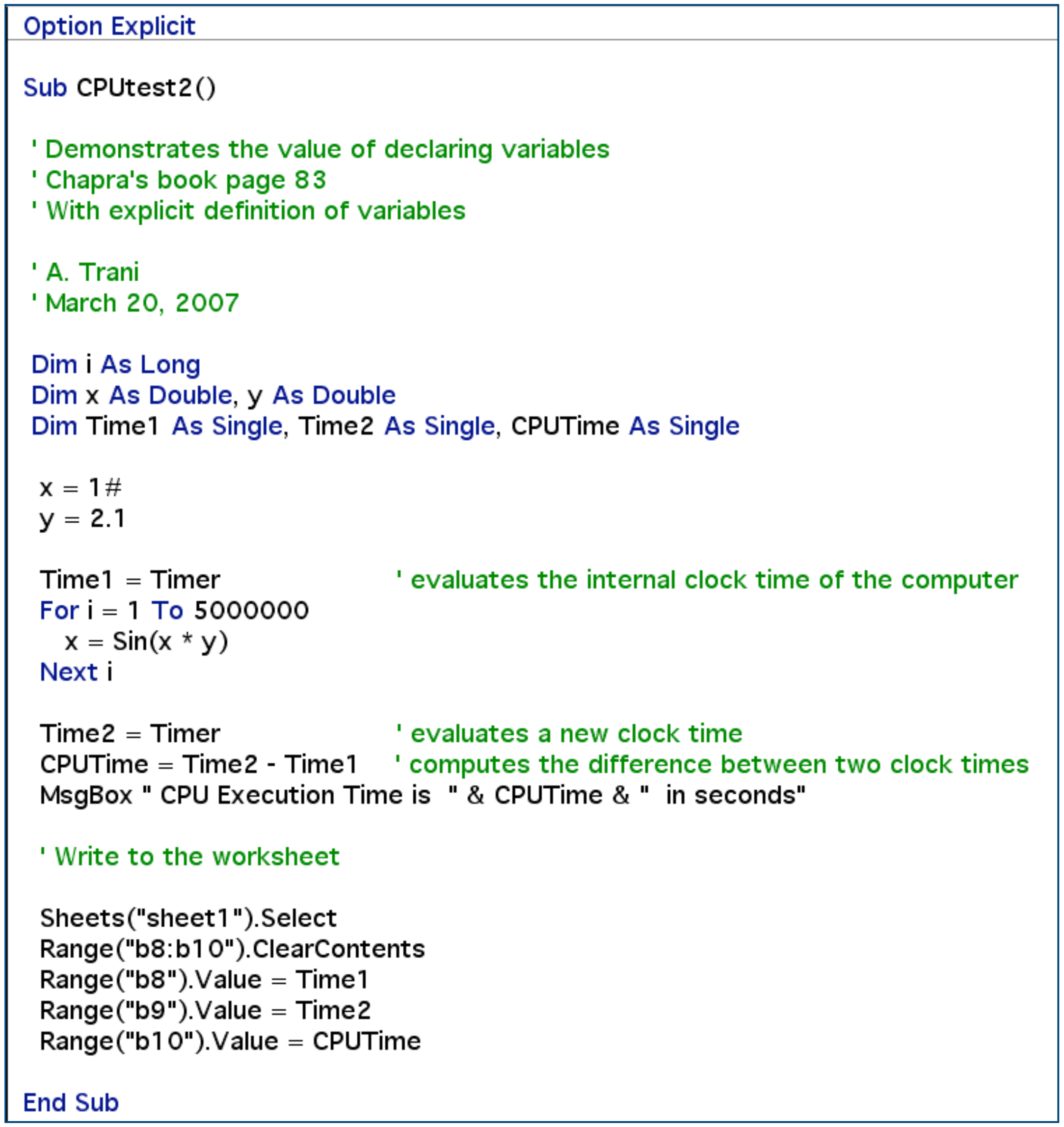
(General) CPUtest

```
Sub CPUtest()

 ' Demonstrates the value of declaring variables
 ' Chapra's book page 83
 ' No explicit definition of variables

 ' A. Trani
 ' March 20, 2007

  x = 1#
  y = 2.1

  Time1 = Timer                  ' evaluates the internal clock time of the computer
  For i = 1 To 5000000
    x = Sin(x * y)
  Next i

  Time2 = Timer                  ' evaluates a new clock time
  CPUTime = Time2 - Time1        ' computes the difference between two clock times
  MsgBox " CPU Execution Time is  " & CPUTime & "  in seconds"

   ' Write to the worksheet

  Sheets("sheet1").Select
  Range("b8:b10").ClearContents
  Range("b8").Value = Time1
  Range("b9").Value = Time2
  Range("b10").Value = CPUTime

End Sub
```

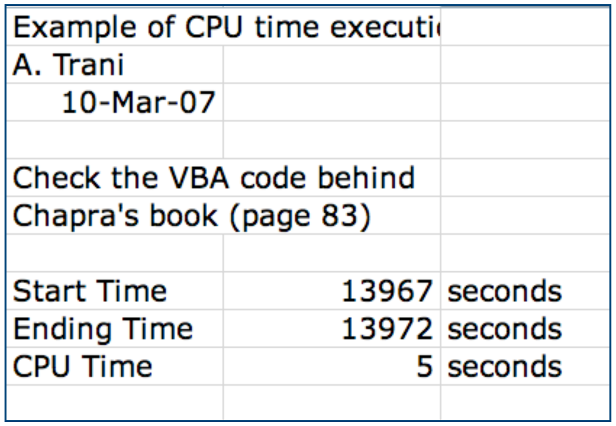


| A | B | C |
|---|---|---|
| Example of CPU time execution in VBA | | |
| A. Trani | | |
| 10-Mar-07 | | |
| | | |
| Check the VBA code behind | | |
| Chapra's book (page 83) | | |
| | | |
| Start Time | 14123 | seconds |
| Ending Time | 14130 | seconds |
| CPUTime | 7 | seconds |

# Example (with Variable Declarations)



```
Option Explicit

Sub CPUtest2()

 ' Demonstrates the value of declaring variables
 ' Chapra's book page 83
 ' With explicit definition of variables

 ' A. Trani
 ' March 20, 2007

 Dim i As Long
 Dim x As Double, y As Double
 Dim Time1  As Single, Time2 As Single, CPUTime As Single

  x = 1#
  y = 2.1

  Time1 = Timer                    ' evaluates the internal clock time of the computer
  For i = 1 To 5000000
     x = Sin(x * y)
  Next i

  Time2 = Timer                    ' evaluates a new clock time
  CPUTime = Time2 - Time1          ' computes the difference between two clock times
  MsgBox " CPU Execution Time is  " & CPUTime & "  in seconds"

  ' Write to the worksheet

  Sheets("sheet1").Select
  Range("b8:b10").ClearContents
  Range("b8").Value = Time1
  Range("b9").Value = Time2
  Range("b10").Value = CPUTime

End Sub
```



| | | |
|---|---|---|
| Example of CPU time executi | | |
| A. Trani | | |
| 10-Mar-07 | | |
| | | |
| Check the VBA code behind | | |
| Chapra's book (page 83) | | |
| | | |
| Start Time | 13967 | seconds |
| Ending Time | 13972 | seconds |
| CPU Time | 5 | seconds |
| | | |

## Common Errors

1. Arithmetic operations cannot be implied

   $z = 10(x_1+2x_2)$ ` illegal

   ```
   z = 10*(x1+2*X2)
   z = 10(x1+2x2)     ` illegal
   ```

2. Two or more operators cannot occur consecutively

   ```
   z = x + -y
   z = x + (-y)
   ```

3. There must be an equal number of left and right parentheses

4. Arithmetic expression must accurately represent original sequence of calculations

5. Variables appearing on right side of arithmetic expression must be assigned values prior to their use.

## Mixed Mode Arithmetic

- BASIC converts all operands in the statement so they have the same precision as the most precise operand

```
z = 3/4          0.75
a% = 200/3       67
a = 200/3        66.66666
a# = 200/3       66.66666666666667
a# = 200#/3      66.66666666666667
```

- In integer division operands are rounded to integers <u>before</u> division and result are <u>truncated</u>

```
a = 8.4/9        0
a = 8.99/9       1
a = 8/3          2
```

## Comments

- A well written program is easy to understand by both the programmer and users
- Comments are non-executable statements with means that the computer ignores these statements during program execution
- Comments enhance make your program more readable
- Comments are an invaluable part of your program
- They assist you and others in understanding the logic of the program

```
Rem   -- This is a comment
` This is also a comment
```

## Effect of Declarations in Functions and Subroutines

CEE 3804 Computer Applications for CEE

## Modular Programming

- A consequence of top-down design is that the problem is decomposed into smaller and simpler sub-problems
- The program is broken up into a number of smaller subprograms or ***modules***

- This approach of designing programs as a series of modules is called ***modular design***
  –
- A module is a small self-contained section of an algorithm
- Modular design has a number of advantages

## Advantages of Modular Design

- Modular programs are easier to write and debug
- Each module can be written and tested independently
- Modular programs are easier to debug
- Modules can be changes, rewritten or even replaced
- Previously developed and tested modules can be used in different programs
- Can develop a library of modules

## Visual Basic Modules

- In Visual Basic a module is a source file which can contain one or more procedure
- A procedure is either a subroutine or a function
- A module file can contain one or more procedures
- The procedures in a module are global which means that they can be invoked from anywhere in the program

## Recall Procedures in VBA

- There are two types of procedures (subprograms) in Visual Basic
- Subroutines
  - Sub .. .. End Sub
  - Can return zero or more values. Cannot be used in as expression
- Functions
  - Function .. .. End Function
  - Can return only one value. Can be used in expressions. Similar to the Visual Basic built-in functions such as Abs, Log, Sin

## Functions

- A function consists of a block of instructions that begins with the Function statement and ends with an End Function
- Functions are invoked the same way as Visual Basic built-in functions
- You specify the name of the function in expressions, as arguments in statements or other functions
- Can use a function in any place you can use a built-in function
- Functions can only return one value

# Functions

- **Syntax**

```
Function functionName (parmlist)
   .. ..
   .. ..
   .. ..

   functionName = .. .. ..

End Function
```

*assign value to functionName in VBA. In VB, can use return()*

## Functions

- Parameter list contains variables that will receive values when the function is called
- These variables are also called formal parameters
- In VBA, must have an assignment statement in the function body that assigns a value to functionName
- The value assigned to functionName is the value returned from the function
- In VB, use the return() function, e.g. return(x) will cause the function to return the value of x

## Functions

- **General Format**

```
[Static][Private] Function funcName [parmlist] [As type]
   funcName = expression
End Function
```

funcName is the name of the function

parmlist is the list of formal parameters

type specifies the type of value

expression is any general expression that has the same data type as funcName

## Functions

- If no data type is defined, the default is Variant in VBA, object in VB

```
Function Square (x as Single) As Single
   Square = x * x
End Function
```

- Function Square can be called as follows:

```
y = Square(x)
y = Square(10)
z = y * Square(x)
```

## Preferred Approach

```
Function fName (parm1 As type, parm2 as type) As type
```

- Should explicitly indicate the type of each of the formal parameters in the function
- Should explicitly indicate the type of value returned by the function
- Should include comments near the top to indicate what the function does, and also what it returns
- This makes it easier to understand exactly what the function does and how it is to be used

## Example of a Function

```
Function MaxOf3(a as Single, b as Single, c As Single) As Single
`-----------------------------------------------------
` Purpose:
`  Determine the largest of three numbers
` Input Parameters:
`   a - first number
`   b - second number
`   c - third number
` Returns:
`   The largest of a, b an c
` ----------------------------------------------------
    Dim max as Single
    If a > b And a > c Then
       max =  a
    Else If b > c Then
       max = b
    Else
       max = c
    End If
    MaxOf3 = max
 End Function
```

## Invoking Functions

- Functions can be used in expressions just like variables. Consider the function

```
Function Log10(x as Double) as Double
```

- This function can be called as

```
y = Log10(3.4)
y = 2.3 * z * Log10(w)
y = a + Log10(b)
```

- The function MaxOf3 can be called as

```
x = MaxOf3(a,b,c)
w = z + MaxOf3(2.0, 50.0, x )
y = MaxOf3(2. * x, a, 4./z)
```

## Subroutines

- Block of instructions that begins with a Sub statement and ends with an End Sub statement
- When a Sub procedure is called, control is transferred to the subroutine and instructions within the subroutine are executed
- Control is transferred back to the calling program when an End Sub statement or an Exit Sub statement is executed

## Subroutines

- Syntax

```
Sub subName (parmlist)
    ......
    ......
    ......
End Sub
```

- Parameter list contains variables that will receive values from the calling program
- These variables are also called formal parameters
- Items in the parameter list are separated by commas
- Some of these variables are input variables while others are output variables.

## Declaring a Sub Procedure

- General syntax

```
[Static][Private] Sub subName[parmlist]
   instructions
[Exit Sub]
    instructions
End Sub
```

subName is the name of the Sub procedure

parmlist is the list of formal parameters

instructions is a block of Visual Basic instructions

## Example of a Subroutine

```
Sub MaxOf3(a as Single,b as Single,c As Single, max As Single)
`-----------------------------------------------------
` Purpose:
`  Determines the largest of three numbers
` Input Parameters:
`   a - first number
`   b - second number
`   c - third number
` Output Parameters:
`   max - the largest of a, b an c
` ---------------------------------------------------
    If a > b And a > c Then
       max =  a
    Else If b > c Then
       max = b
    Else
       max = c
    End If
End Sub
```

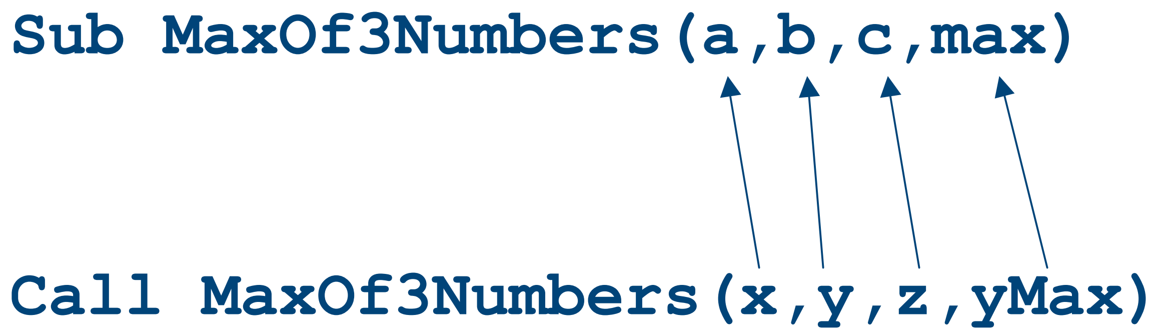
## Calling Subroutines

```
Call subName (argument list)
```

- Call statement transfers control from calling program to subroutine
- Argument list specifies the variables that are passed to the subroutine

## Calling Subroutines

- There is a one-to-one correspondence between variables in the parameter list and arguments in the argument list



```
Sub MaxOf3Numbers(a,b,c,max)


Call MaxOf3Numbers(x,y,z,yMax)
```

a = x, b = y, x = z, max = yMax

## Calling Subroutines

- Arguments in the argument list must correspond in number and type to the parameters in the parameter list
- If they are not in the same order or of the same type, then the values passed to the subprogram from the calling program will be incorrect
- Usually the Visual Basic compiler will give an error message

```
Call MaxOf3Numbers(x,y,z)
Call MaxOf3Numbers(1.,2.,4. “AC”)
```

## Differences Between Subroutines and Functions

- Subroutines cannot be used in expressions
- Subroutines can return zero or more values
- Functions can only return one value
- Functions can be used in expressions
- To call a subroutine we have to use a Call statement in VBA, or simply the subroutine name in VB

## Differences Between Subroutines and Functions

- Parameter list in functions works the same as for subroutines
- Static and Private keywords work the same as they do with subroutines
- Exit Function is similar to Exit Sub instruction
- Main difference is that a function assigns a value to the name of the function itself
- To call a function we do not use a call statement
- We invoke a function in the same was as we do one of the Visual Basic built-in functions

## Passing Arguments

- Two fundamental rules for passing arguments

  1. Number of arguments in argument list and the number of formal parameters in the parameter list must both be the same

  2. The data type of each argument and its corresponding formal parameter must match

- The most common mistakes when calling procedures are: too few or too many arguments, or passing arguments that are not of the same type

## Calling Procedures

- There are two ways in which arguments are passed to procedures

1. Call by value

2. Call by reference

## Call By Reference

- When an argument is passed by reference the program passes the address of the variable to the procedure
- The procedure can use this address to access the variable and change it contents
- Call by reference is useful if you need to receive results from the procedure
- You should be careful when using call by reference since the procedure can change the value of the variable which can have adverse side effects in your programs

## Call by Value

- When an argument is passed by value the compiler makes a copy of the argument and then passes this copy to the procedure
- The procedure can change the value of the argument but any changes it makes affects the local copy and not the original variable
- Call by value is useful for sending “input” arguments to a procedure
- There are several advantages to call by value
- Errors are localized since changes are made to the copy
- Side effects are minimized

## Passing Arguments

- The default for VBA is call by reference
- In VB and VB.Net use the ByVal or ByRef to pass by value or reference respectively.

```
Call MaxOf3(byval x as integer, byval y as
  integer, byval z as integer, byref max as
  integer)
```

  - In the above example, x, y and z are passed by value.
  - The subroutine cannot change the value of x, y, and z.
  - The variable max is passed by reference since the subroutine needs to change the value of max.

## Passing Arguments By Value

- When the actual argument is a literal, a constant or an expression, parameter passing is by value
- In this case, Visual Basic passes the value of the parameter rather than the address
- The value of the expression is calculated, the result is stored in a temporary location and the address of this temporary location is passed to the procedure

## Passing Parameters By Value

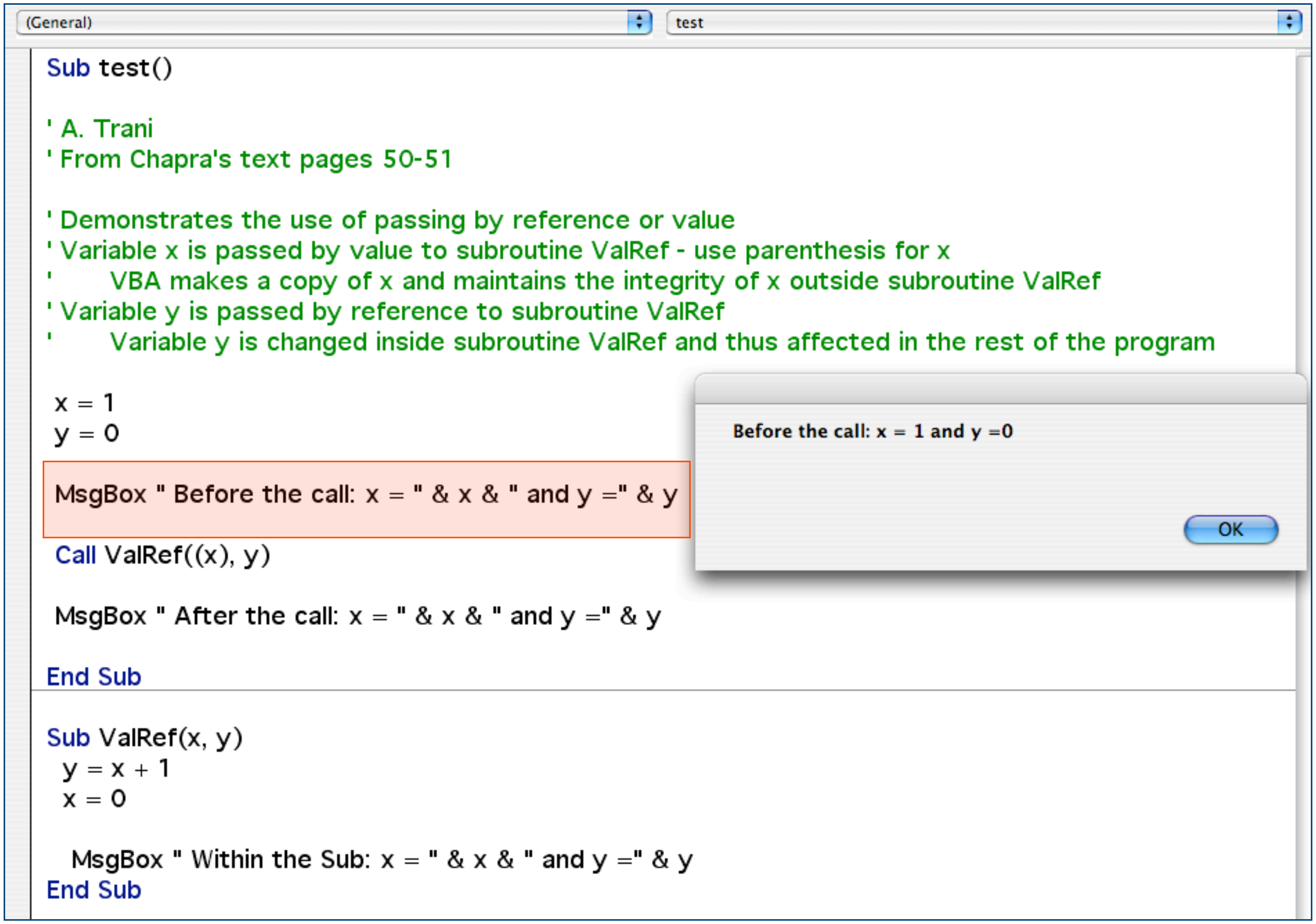
- You can pass parameters by value using parenthesis for the variable in question
- `Function Square ((x) as Single) As Single`
- The parameter x is now passed to the procedure by value

- If you use VB 6.0 or VB.Net
- In VB 6.0 or VB.NET you place the ByVal keyword before the formal parameter in the procedure declaration
  - `Function Square (ByVal x as Single) As Single`
- The parameter x is now passed to the procedure by value

```
Sub MaxOf3Numbers(ByVal a as Single, ByVal b as Single,
   ByVal c as Single, byref max as Single)
```

The parameters a, b and c are now passed to the procedure by value

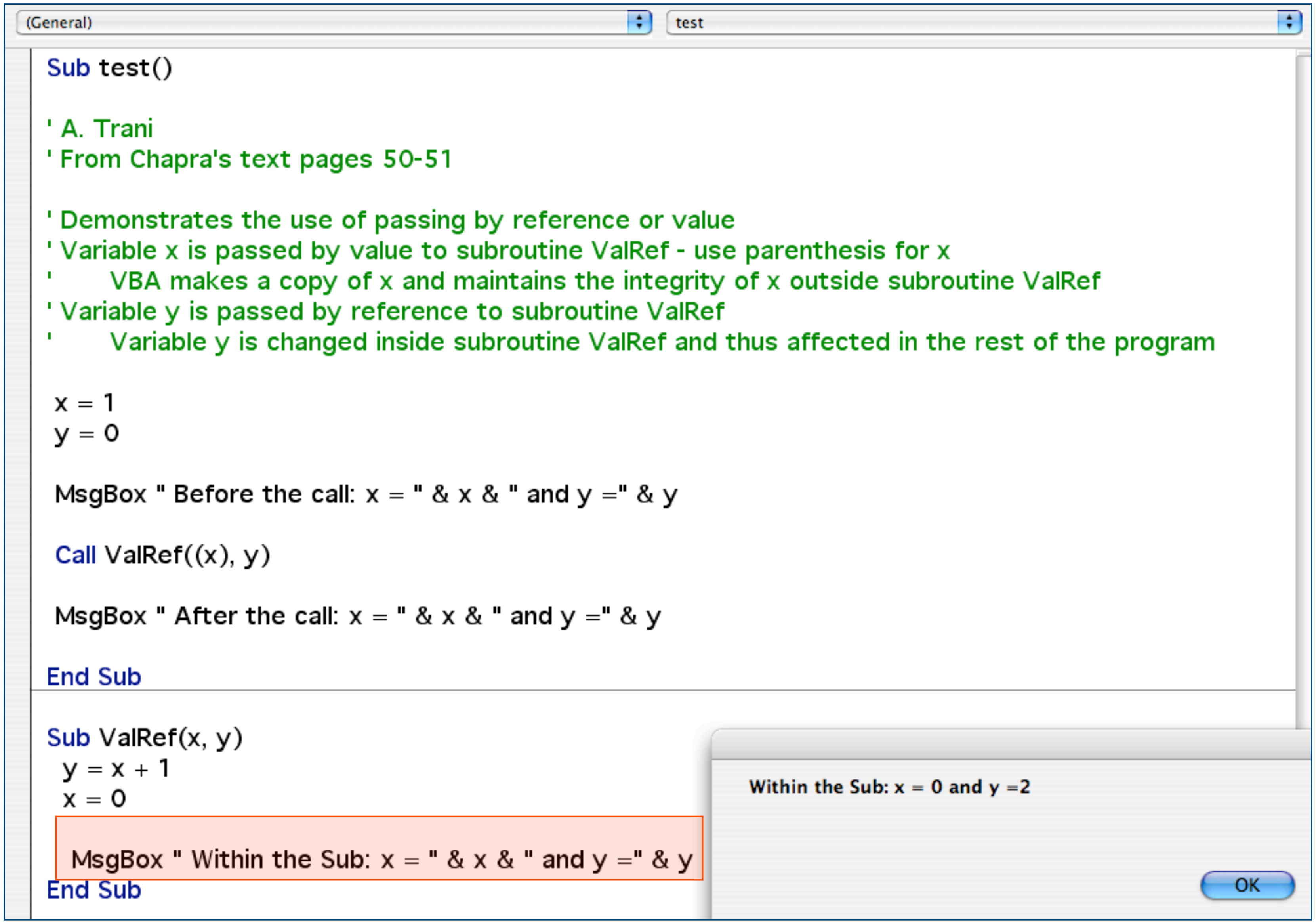
## VBA Example (Chapra’s Book pages 50-51)



```
Sub test()

' A. Trani
' From Chapra's text pages 50-51

' Demonstrates the use of passing by reference or value
' Variable x is passed by value to subroutine ValRef - use parenthesis for x
'       VBA makes a copy of x and maintains the integrity of x outside subroutine ValRef
' Variable y is passed by reference to subroutine ValRef
'       Variable y is changed inside subroutine ValRef and thus affected in the rest of the program

 x = 1
 y = 0

 MsgBox " Before the call: x = " & x & " and y =" & y

 Call ValRef((x), y)

 MsgBox " After the call: x = " & x & " and y =" & y

End Sub

Sub ValRef(x, y)
  y = x + 1
  x = 0

   MsgBox " Within the Sub: x = " & x & " and y =" & y
End Sub
```

```
Before the call: x = 1 and y =0
```

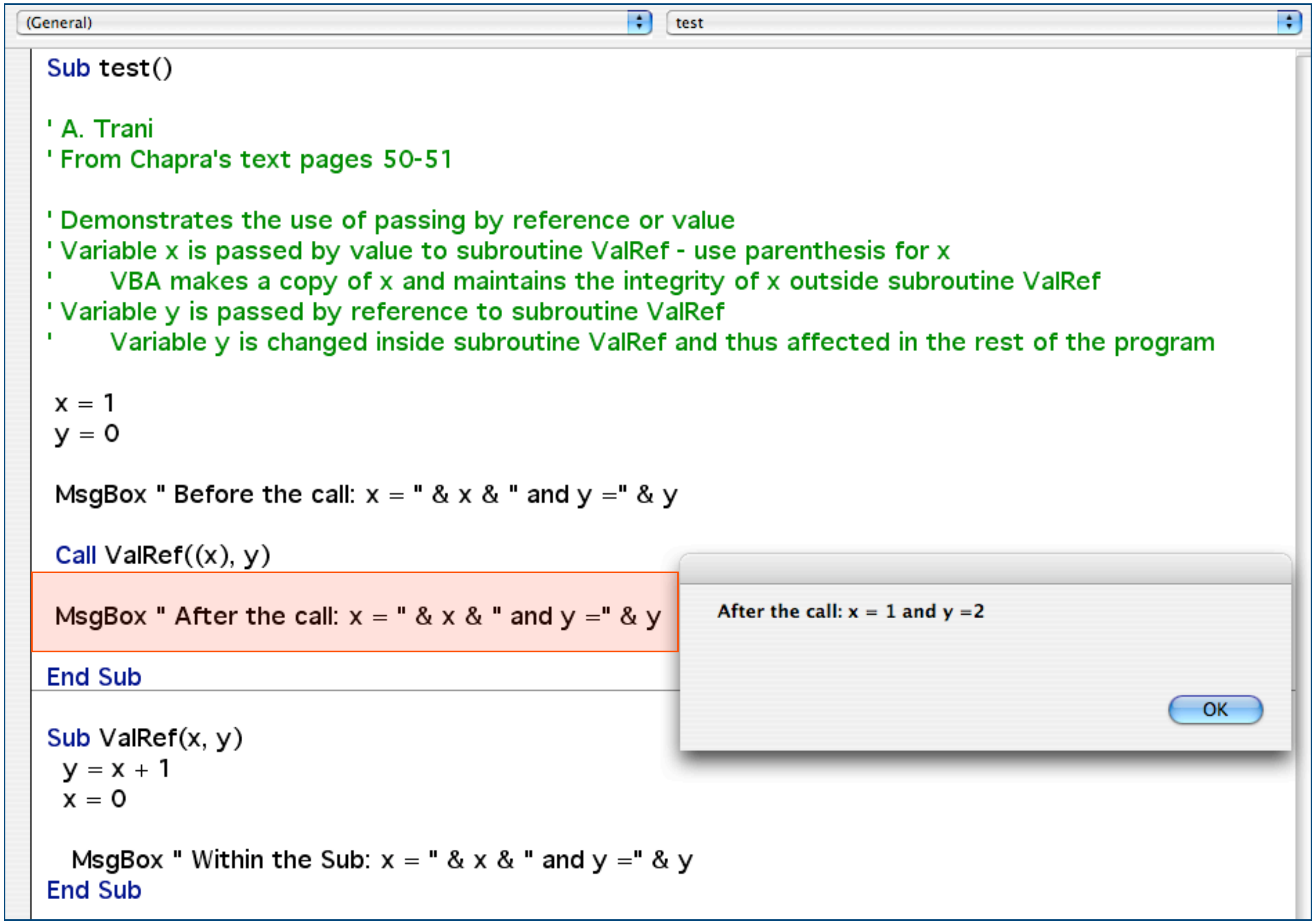
## Example (MsgBox inside Sub ValRef)



```
Sub test()

' A. Trani
' From Chapra's text pages 50-51

' Demonstrates the use of passing by reference or value
' Variable x is passed by value to subroutine ValRef - use parenthesis for x
'       VBA makes a copy of x and maintains the integrity of x outside subroutine ValRef
' Variable y is passed by reference to subroutine ValRef
'       Variable y is changed inside subroutine ValRef and thus affected in the rest of the program

 x = 1
 y = 0

 MsgBox " Before the call: x = " & x & " and y =" & y

 Call ValRef((x), y)

 MsgBox " After the call: x = " & x & " and y =" & y

End Sub

Sub ValRef(x, y)
  y = x + 1
  x = 0

  MsgBox " Within the Sub: x = " & x & " and y =" & y
End Sub
```

```
Within the Sub: x = 0 and y =2
```

# Example (MsgBox after Sub ValRef)



```
Sub test()

' A. Trani
' From Chapra's text pages 50-51

' Demonstrates the use of passing by reference or value
' Variable x is passed by value to subroutine ValRef - use parenthesis for x
'       VBA makes a copy of x and maintains the integrity of x outside subroutine ValRef
' Variable y is passed by reference to subroutine ValRef
'       Variable y is changed inside subroutine ValRef and thus affected in the rest of the program

 x = 1
 y = 0

 MsgBox " Before the call: x = " & x & " and y =" & y

 Call ValRef((x), y)

 MsgBox " After the call: x = " & x & " and y =" & y

End Sub

Sub ValRef(x, y)
  y = x + 1
  x = 0

   MsgBox " Within the Sub: x = " & x & " and y =" & y
End Sub
```

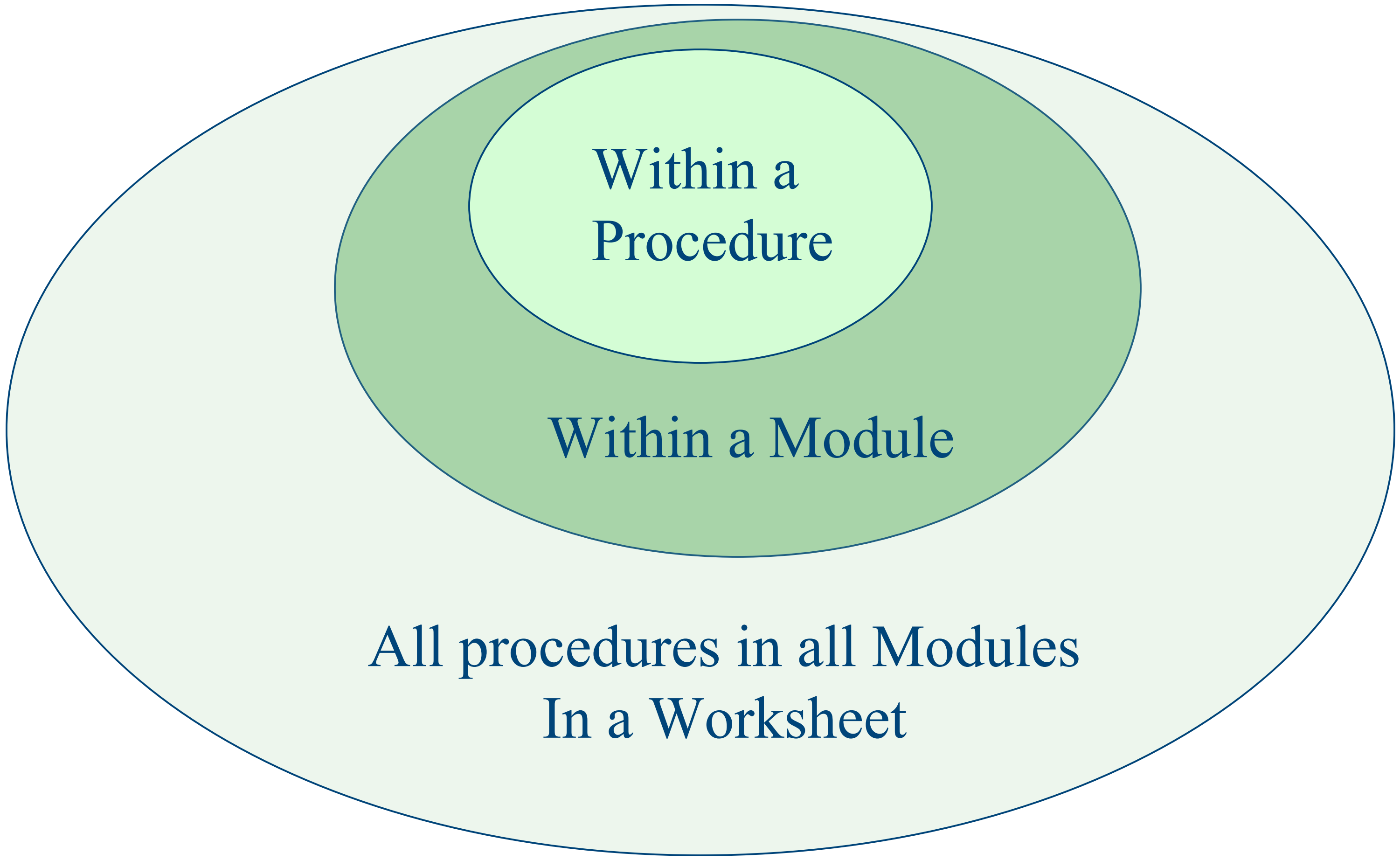
```
After the call: x = 1 and y =2
```

## Storage Class, Scope and Visibility

- There are two attributes associated with each variable
  1. data type
- 2. storage class

- The data type tells the VBA how much memory should be assigned for the variable
- The storage class tells VBA how the variable is to be stored
- The storage class also determines the visibility and scope of the variable

## Levels of Variable Scope



## Scope and Declaration

### Procedure (or local) variables

- Variables declared within a procedure
- Created when procedure starts
- Destroyed when procedure terminates

### Module Variables

- Variables declared at the beginning of a module
- Before the first procedure
- Can be accessed by all procedures within the module
- Created when module starts
- Destroyed when module terminates

### Workbook (or Global) variables

- Variables that can be accessed by all procedures in all modules
- Created using a *Public statement* before the first procedure in a module

## Local Variables

- Variables declared within the body of a function or subroutine are local variables
- Local variables are visible to the procedure but not to other procedures
- Local variables are created when the procedure is executed and are destroyed when the procedure terminates
- Local variables exist only during the duration of the procedure
- Local variables are reinitialized each time the procedure is executed

## Static Variables

- Static variables are permanent variables, that is they remain in existence for the entire duration of the program
- You can define individual variables as static or you can specify that all variables in a procedure to be static

## Static Variables

- To declare individual variables as static you use the following syntax

```
Sub MySub (b as Single,c as Single)
    Static x, y, z as Single
    Static myAddress as String
    Static a as Double
```

- Static variable retain their values between subroutine calls
- Static variables remain in existence for the entire duration of the program

## Static Variables

- Static variables are only initialized once. They are not reinitialized
- To define all variables in a procedure to be static you can use the following syntax

```
Sub MySub(a as Single) As Static
   Dim x,y,z as Single
   Dim c,d as String
```

- All local variables in the subroutine are static and retain their values between calls to the subroutine

## Global Variables

- You should limit the number of global variables in your program
- Global variables are accessible to all procedures
- Programs containing global variables are difficult to maintain and are prone to errors
- Since all procedures have access to global variables an procedure can unknowingly change the value of a global variable leading to problems elsewhere which is difficult to trace.

## Global Variables

- Global variables limit the portability of your code
- If you have a global variable and a local variable of the same name then all references to that variable inside the procedure apply to the local variable and not the global variable

## Public and Private Procedures

- Procedures defined at the form level are available throughout the form
- Procedures defined at the module level are available throughout the application
- Procedures in other modules cannot invoke a procedure declared as Private
- By using Private we can reduce the risk of name conflicts
- Use Public to make a form level procedure accessible outside the form