

The background is a dark blue gradient with a subtle pattern of white dots. Overlaid on the left side are several concentric circles and arcs in a lighter blue color. Some of these arcs have degree markings, such as 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, and 260. There are also small arrows indicating a clockwise direction of rotation.

# PROCEDURES & ARGUMENTS PART 1

VISUAL BASIC

# TABLE OF CONTENTS

1. Lecture context
2. Variables
3. Subroutines
4. Passing Arguments By Value
5. Passing Arguments By Reference

# LECTURE CONTEXT

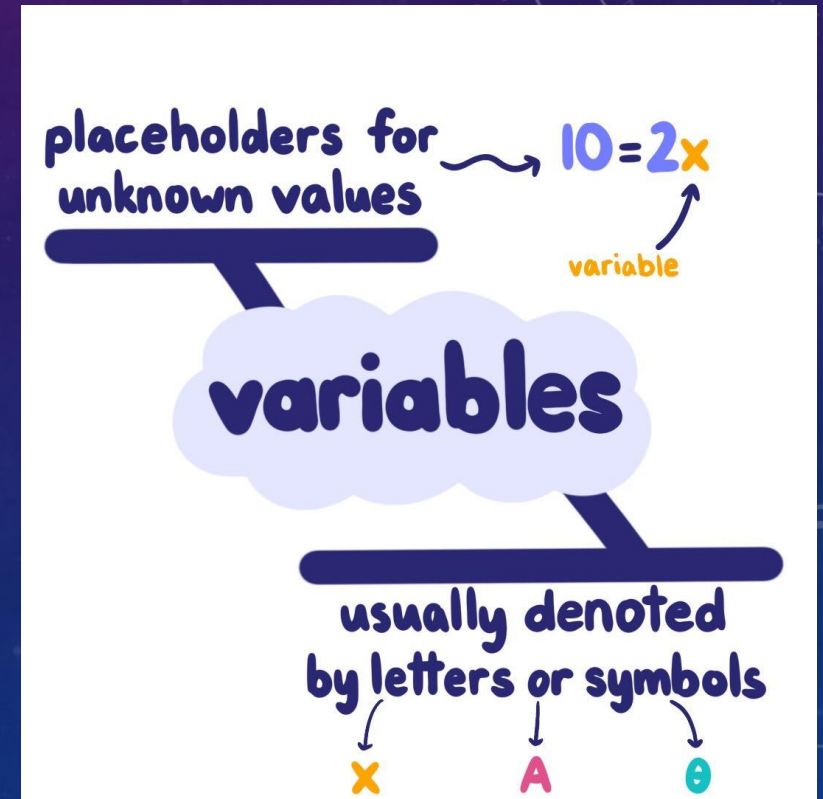
- Most non-superficial programs are very large, with 1000s of lines of code and 100/1000s of variables.
- Programmers need a way to encapsulate portions of a program to develop and test in isolation from the rest of the code.
- It is also important to re-use resources (memory) and control the scope of variables to ensure one does not accidentally change values in other sections of code.
- Much code is written by teams and an individual's section of code also needs to easily integrate into the overall program without modification or issues.
- Procedures, such as Sub Routines facilitate all this.

A large blue circle containing the white text 'VB'. The background of the slide features faint, stylized circular patterns and numbers, suggesting a technical or scientific theme.

VB

# VARIABLES

- Create variables (use memory) when needed + destroy (release memory) when not
- Manage by grouping and *limiting* their Scope to **Procedure Level**
- Global variables **only** used if no other way to share data across modules - never passed!
- Procedures should operate only on Private objects / data passed to them





# SUBROUTINES

- Code within *Sub/End Sub* statements
- private sub *name* (inputs)
- End Sub passes control back to calling program: line after the call is then run

```
Private Sub DisplayCount()  
    intCount = intCount +1  
    txtDisplay.text = intCount  
End Sub
```

# SUBROUTINES

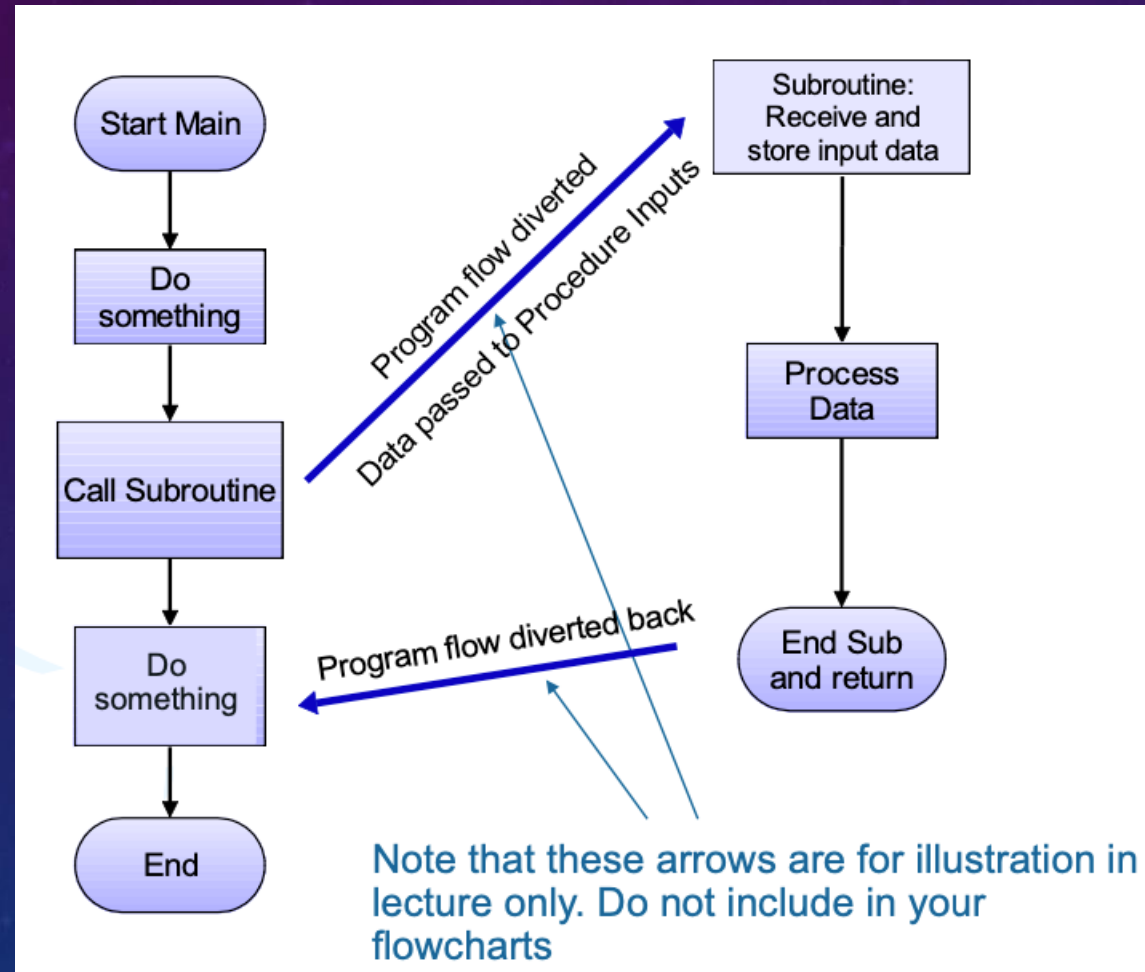
- Invocation by name: *Call DisplayCount*
- May pass data into Sub using brackets with 'Call' or none without Call command
- *Call FindSum(23, 34)* or *FindSum 23, 34*
- Event handlers are Subs - may be called
- Program flow immediately diverted to sub
- 'End Sub' returns program flow to line following the call in original routine
- In standard module use Public sub *name()*

*Private Sub DisplayCount()*

*intCount = intCount +1*  
*txtDisplay.text = intCount*

*End Sub*

# HOW DOES A SUBROUTINE WORK ?



# PASSING ARGUMENTS BY VALUE

- Arguments usually = data (constant or variable)
- Passes data values to procedure but does not give the procedure access to their address
- Procedure gets copy of original variable contents
- If the procedure changes the value, the change affects only the copy and not the variable itself
- Use the **ByVal** keyword to indicate an argument passed 'by value'





# PASSING BY VALUE EXAMPLE

- This Subroutine receives copies of two DATA with no link to original data
- These receive and store data for processing

```
Private Sub Addition(ByVal intPassed1 As Integer, ByVal intPassed2 As Integer)
    Dim Add As Integer

    Add = intPassed1 + intPassed2
    intPassed1 = 0
    intPassed2 = 0
    txtResult.Text = Add

End Sub
```

Input variables declared in the brackets

# PASSING BY VALUE EXAMPLE

- Passes copy of CONTENTS of *intInput1* & *intInput2*
- After run: *intSum* = 12 , *intInput1* = 10, *intInput2* = 2
- **NOTE** No change made to original variables' contents from inside the calling procedure, only changes are to those variables within the Sub

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Dim intInput1 As Integer, intInput2 As Integer

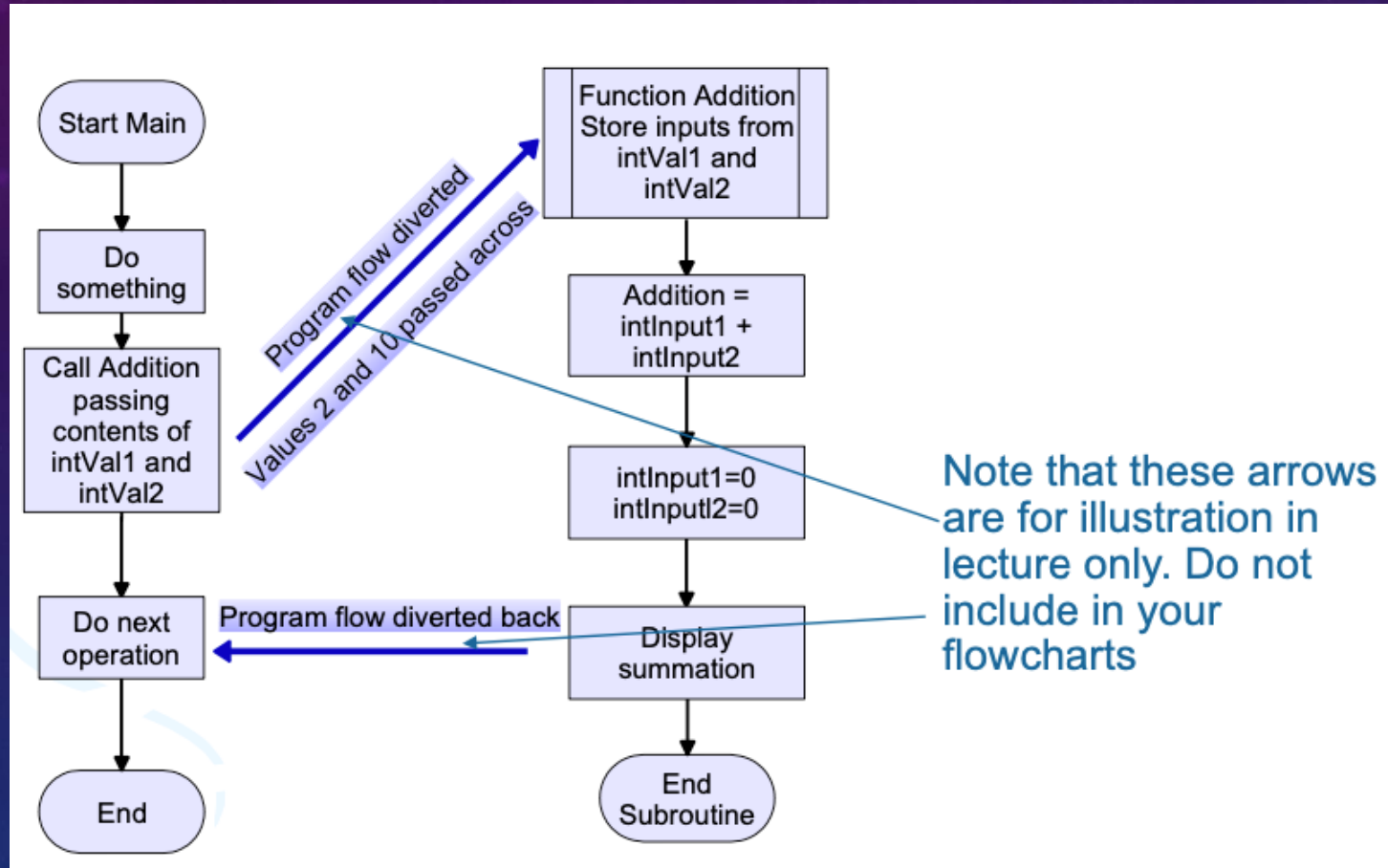
    intInput1 = txtNumber1.Text
    intInput2 = txtNumber2.Text

    Addition(intInput1, intInput2)

    txtOrigVars.Text = intInput1 & " and " & intInput2
End Sub
```

The screenshot shows a Windows Form titled "Form1" with a green button labeled "Pass Values". Below the button, the text "Sum result" is displayed above a text box containing the value "12". Further down, the text "Variable contents after call" is displayed above two text boxes. The first text box contains the value "2", and the second text box contains the value "10 and 2".

# BY VALUE FLOW ILLUSTRATION



# PASSING ARGUMENTS BY REFERENCE

- An invoked Procedure given access to original variable's address in RAM: link to the address
- Any changes made to passed argument in the invoked procedure **affects the original variable contents too**
- Data type for an argument passed by reference must match type in the Procedure *declaration*
- *Good practice to use different names for data in sending procedure and receiving procedure*





# PASSING BY VALUE EXAMPLE

- This Subroutine receives copies of two DATA with no link to original data
- These receive and store data for processing

```
Private Sub Addition(ByVal intPassed1 As Integer, ByVal intPassed2 As Integer)
    Dim Add As Integer

    Add = intPassed1 + intPassed2
    intPassed1 = 0
    intPassed2 = 0
    txtResult.Text = Add

End Sub
```

Input variables declared in the brackets

# PASSING ARGUMENTS BY REFERENCE

- This Subroutine receives reference to two **ADDRESSES** = links to original data
- Any changes made to passed argument in the invoked procedure **affects the original variable contents too**
- Data type for an argument passed by reference must match type in the Procedure *declaration*
- *Good* practice to use different names for data in sending procedure and receiving procedure

```
Private Sub Addition(ByRef intPassed1 As Integer, ByRef intPassed2 As Integer)

    Dim Add As Integer

    Add = intPassed1 + intPassed2
    intPassed1 = 0
    intPassed2 = 0
    txtResult.Text = Add

End Sub
```

put variables declared ByRef which is the only change to the previous ByVal program

# PASSING ARGUMENTS BY REFERENCE

- Passes ADDRESSES of *intInput1* & *intInput2*
- After run: *intSum* = 12, *intInput1* = 0, *intInput2* = 0
- NOTE original variables' contents were set to ZERO by the sub routine's code setting its local variables to zero because of the addresses being passed

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Dim intInput1 As Integer, intInput2 As Integer

    intInput1 = txtNumber1.Text
    intInput2 = txtNumber2.Text

    Addition(intInput1, intInput2)

    txtOrigVars.Text = intInput1 & " and " & intInput2
End Sub
```

The screenshot shows a Windows Form titled "Form1" with a green button labeled "Pass Values". Below the button, the form displays the results of a calculation. The "Sum result" is shown as 12. The "Variable contents after call" are shown as 2 and 0 and 0.

Pass Values	
10	12
Sum result	
2	0 and 0
Variable contents after call	