**5.05 Looping Structures**

Computer Programming I

Essential Standard: 5.00 Apply Programming and Conditional Logic

**Getting Input from an InputBox**

* + Used to get Input from the User
  + We already learned how to do this using a text box.
  + There is another method: the input box.
  + The **input box** is a dialog box that pops up and prompts the user for input. The program stops and waits until the user inputs data or cancels the box.
  + **Input Boxes**
  + Syntax: StrVariable = InputBox (prompt, title)
  + The prompt and title can be string literals or a string variables.
    - String literal is “String Here”
    - String Variable is strVariable that has already been declared.
  + **Uses**:
    - **Repeated Data Input**: Input boxes can be placed within a **loop** and provide input repeatedly until the user ends the loop. (Ex: User keys 25 Sales into an inputbox and program sums them as they come into the loop)
    - **“Empty” Data Validation**: Input Boxes can check that the user actually keyed something. Code uses the **NOTHING** keyword and an **IF** statement with the InputBox. (Examples below).
  + We can check that the user actually inputted something with the **Nothing** keyword and an IF statement.
  + Method 1 continues to ask for input on every use until user puts it in:

If strName = Nothing Then

strName= InputBox(“Enter your name to continue!:”, “What is your name?”)

End If

* + Method 2 ends program if empty:

If strName = Nothing Then

Messagebox.Show(“Bye!”)

Application.Exit()

End If

**Val() Function**

* + Val() returns a numeric version of a string number or returns a **zero** if there is no number.
  + Use when assigning string data into a numeric variable. (from a textbox, an inputbox or a string variable).
  + Examples:
  + dblSale = **Val**(txtSale.Text)
  + dblSale = **Val**(InputBox(“Enter cost of the sale”, “Bill”)
  + dblSale = **Val**(strSale)
  + Used only in Visual Basic language.
  + Instead of Convert.ToDouble or Convert.ToInt32
  + Val() Function & InputBoxes can be used together for Data Conversion from String to Numeric as it is entered

**Special Variables with Loops**

* + Counter Variables
    - When dealing with loops you also may use a counter variable to determine the number of times a loop executes.
    - This type of counter you can declare Dim instead of Static outside of the loop.
    - This variable (normally of integer data type) helps the computer know where it is in the loop.
    - The most common name for a counter variable is i. Next comes j, then k, and so on.
    - Increment the Counter
    - After each run through the loop we want to add (or subtract) some number (usually 1) from the counter variable.
    - Counter Examples
    - i = i + 1 or i+=1

i = i - 1 or i-=1

i+=2 or i-=2

* Accumulator Variables
  + Similar to a counter, except the value that updates the accumulator changes
  + A variable storing a number that is incremented by a ***changing*** value.
  + Syntax:

accumulator = accumulator + value

accumulator += value

* + Useful for keeping a running total

**Looping Structures**

* + **Looping** = Repeated Actions
  + Have you ever performed the same action over and over?
    - For example: As long as it rains I am going to sing the same song. When the song finishes I am going to start over again.
  + Programmers can code repeated actions into their programs. This is called a **loop.**
  + Each time the loop runs is called an **iteration**.

**Types of Loops**

* + These are the four types of loops in VB. We will cover each in detail.
    1. Do While
    2. Do…Loop While
    3. For…Next
    4. For Each…Next
       - The For Each loop will be covered later.

**Pretest and Posttest**

* + When dealing with loops there are two different ways to test a condition.
  + **Pretest**
    - Test the condition BEFORE the loop runs. If the condition is false the loop will not execute.
    - A pretest loop only executes if the condition is true for at least one time.
  + **Posttest**
    - Run the loop one time then test the condition- if the condition is false the loop will then terminate.
    - A posttest loop will ALWAYS execute at least one time.

**Do While…Loop**

* + Syntax

**Do While** condition ‘The condition must be a Boolean Expression.

*Statements*

**Loop**

* + This form executes only if condition is true, therefore if condition is false, the loop does not execute.
  + This is a pre-test loop.
  + Example

sum = 2  
**Do While** sum < 10 ‘The condition must be a Boolean Expression.  
 sum += 2  
**Loop**

* The statement does iterate (loop) until sum is 10 or greater.  
  + Example
    - Add all even numbers 1 to 25 - skip all odd numbers.

Dim i As Integer = 0

Dim intResult As Integer = 0

Do while i < 25

intResult = intResult + i ‘updates intResult – This is an accumulator.

i+=2 ‘increases i by 2 – This is a counter.

Loop

* You can use a Do While Loop to Validate Input

Dim intNum As Integer = -1

Dim strInputNum As String = "“

*‘This input “primes” the strInputNum so that is has a starting value.*

strInputNum = InputBox("Enter a Number between 1 & 10", "Number")

*‘The condition will first check to see if nothing was entered, then it checks to see if*

*‘what was entered is less than 0 or greater than 10. If either is true, then it enters  
 ‘ the loop*

Do While strInputNum = Nothing Or (Val(strInputNum) < 0 Or Val(strInputNum) > 10)

*‘The if checks the same condition, if true, then the MessageBox is shown and  
 ‘input is asked for again through the InputBox.*

If strInputNum = Nothing Or (Val(strInputNum) < 0 Or Val(strInputNum) > 10) Then

MessageBox.Show("Enter a Number between 1 & 10")

strInputNum = InputBox("Enter a Number between 1 & 10", "Number")

End If

Loop

*‘Once input is good, this MessageBox is displayed.*

MessageBox.Show("You entered a number between 1 & 10. Your number was " & strInputNum)

**Do…Loop While**

* + The looping structure that evaluates a condition after executing a loop once.
  + Syntax

**Do**

*Statements*

**Loop While** *condition*

* + - *Statements* is the loop and is executed at least once.
    - *Condition* is a Boolean expression used to determine if the loop is to be repeated.
    - *condition is true* 🡪 *repeats*
  + The looping structure that executes a set of statements as long as a condition is true.
  + The condition is a Boolean expression.
  + Evaluates to T or F
  + Executes at least once.
  + The loop below iterates (repeats) while sum is less than 10:  
       
     sum = 0;  
     **Do**   
     sum = sum + 2  
     **Loop While** sum < 10
  + Do…Loop While Example
    - Write a program that adds all numbers up from 1 to 50.

Dim i As Integer = 0

Do

intResult = intResult + i ‘This is an accumulator

i += 1 ‘This is a counter

Loop While (i <= 50)

lblResult.Text = intResult.ToString()

* + Using the Do..Loop While to Validate Input

Dim intNum As Integer = -1

Dim strInputNum As String = ""

Do

If strInputNum = Nothing Or (Val(strInputNum) < 0 Or Val(strInputNum) > 10) Then

MessageBox.Show("Enter a Number between 1 & 10")

strInputNum = InputBox("Enter a Number between 1 & 10", "Number")

End If

Loop While (strInputNum = Nothing Or (Val(strInputNum) < 0 Or Val(strInputNum) > 10))

MessageBox.Show("You entered a number between 1 & 10. Your number was " & strInputNum)

**For Next Loop**

* + A looping structure that executes a set of statements a fixed number of times.
  + Executes until counter is reached.
  + The counter is automatically incremented by 1.
  + Syntax:

**For *counter* = *start* To *end***

*Statements*

**Next *counter***

* + - Where *counter, start and end are* ***Integer*** *variables.*
  + How the For…Next Works
    - You create an **integer counter** to use after the keyword **FOR**.
    - The **first time** the For line executes, your **counter is set** to whatever is after the “=”. It does NOT reset each time the loop repeats.
    - **Each time “Next”** is executed, the **counter is updated by 1** (default value is 1)
    - The **condition is tested** on the **FOR line**
    - The condition is **still true** when counter reaches the **value after** the “**TO**”. (Loop still executes)
    - When the **condition** is tested **false**, execution **jumps** to statement **after** the **NEXT**.
    - Remember, counter is updated to false value and holds that value when you exit the loop.
* For…Next Example
  + - The loop below executes until intNum is equal to 5, by checking one last time (it is no longer true), jumps to Next and exits the loop.

**For** intNum **=** 1 **To** 4

sngTotal = sngTotal + intNum

**Next** intNum

* + - The variable counter “intNum” holds a 5 after the loop ends, but the code inside only executed four times.
* For…Next Internal Counter
  + - You may create the counter variable (intCount) in the **For** line by using the optional **As Integer** keywords, rather than using a **Dim** command before the loop.
    - The lifetime of the variable counter created this way is the lifetime of the loop. (When you exit the loop, the variable counter no longer exists).

**For** intCount **As Integer =** 0 **To** 4  
 sngTotal = sngTotal + intCount  
 **Next** intCount

* The For … Next Statement with Step
  + Step
    - Changes the way the counter is incremented.
    - Can be positive or negative
  + Syntax

For counter = start To end **Step** stepnum

*Statements*

Next counter

* + For…Next…Step Example – Stepping Negative (Backwards)
    - The step integer can be a negative number to tell the compiler to decrement the counter.
    - Make sure your start is the high number, the end is the low number.

For intY = 5 To 1 Step -1

MessageBox.Show("Counting Down " & intY)

Next intY

* + For…Next…Step Example
    - Add all even numbers 1 to 250 - skip all odd numbers.

For i as Integer = 0 to 250 **Step 2**

intResult = intResult + i ‘This is an accumulator

Next i

lblResult.Text = intResult.ToString()

* + - **Step** is a function in VB that changes how the counter will increment. Now we are counting by 2’s instead of 1’s.

**Special Variables with Loops**

* + Using Flags
    - Flag = Sentinel
    - A condition used to signify that a loop should stop executing.
    - Example 1

**Const strFLAG As String = “**Quit**”**

strTempInput = **InputBox**(“Enter a positive number (Quit to stop)”)

**Do While** strTempInput **<> Nothing Or**

strTempInput **<> strFLAG**

*Statements*

**Loop**

* + - Example 2

Const intFLAG As Integer = -1

strTempNum = InputBox(“Enter a positive number (-1 to stop)”)

Do While strTempNum <> Nothing Or strTempNum <> intFLAG

Statements

Loop

* + Flag, Counter & Accumulator Example

Const strPROMPT As String = “Enter Number, use “STOP” to end”

strTempNum = **InputBox (**strPROMPT**,** TITLE**)** ’Input once before ‘loop

**Do While** strTempNum <> “STOP”

num = Convert.ToInt16(strTempNum)

numcounter += 1 ’Counter

sum += num ’Running total

strTempNum = **InputBox** (strPROMPT, TITLE) ’Repeated input ‘ for each iteration

**Loop**

* + Endless Loop
    - A logic error known as the endless or infinite loop occurs when the programmer forgets to increment the counter variable.
    - A loop is endless when it is always true - so it will run forever until the computer crashes (or modern operating systems/web browsers will alert the end user so they can terminate the program).  
       Dim x As Integer = 5  
       Do While (x < 10)  
       x -= 1  
       Loop

**So Which Loop to Use?**

* + Use a For…Next loop when you **know how many times** the loop will run. This can be in the form of constants or variables. You might not know the numbers, but they can be input right before the For…Next runs, ie you can ask the user for this. In a FOR…Next loop, **False** **happens automatically.** It is based on the setup of the For line and the automatic incrementing of the counter variable when the Next line executes (default without STEP adds 1 to counter on Next line).

* + Use a While loop (or Do While) when **your logic TURNs the condition false.** This can be a variable used in the condition that changes based on the program logic, or a Flag entered by the user. In a While loop, logic must create the “False” condition. If you don’t make it happen, an infinite loop results.
    - Example: Do While i < intGrade
    - Example: While(i<intGrade)

**Programming Assignment**

Write a program has 3 buttons- one for each loop type. (Do, While, and For)

* + Each loop should add all even numbers one through fifty and write the result to a label.
  + The result should be 650.

Solution

Private Sub btnFor\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnFor.Click

Dim intSum As Integer

For intNum As Integer = 0 To 50 Step 2

intSum += intNum

Next

Me.lblForAnswer.Text = intSum

End Sub

Private Sub btnDo\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnDo.Click

Dim intNum, intSum As Integer

Do

intSum += intNum

intNum += 2

Loop While (intNum <= 50)

Me.lblDoAnswer.Text = intSum

End Sub

Private Sub btnDoWhile\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnDoWhile.Click

Dim intNum, intSum As Integer

Do While (intNum <= 50)

intSum += intNum

intNum += 2

Loop

Me.lblDoWhileAnswer.Text = intSum

End Sub