**Computer Programming I**

**Essential Standard: 7.00 Apply Advanced Logic**

**Indicator: 7.01 Apply sub procedures/methods and user defined functions.**

Breaking it Up

* As programs get more complex they become longer and harder to debug or even figure out what exactly is going on.
* Sub Procedures provide a way to break up long coding segments and reduce repeated code.
* The breaking up of a program is called modularization because it breaks the program up into modules.

Advantages

* Divides a program into smaller, more manageable parts
* The problem becomes a set of smaller problems
* Easier to understand
* Less code redundancy
* Reuse code
* More flexible
* Use code when it is needed.

Sub Procedure versus Event Procedure

* So far we have been creating Event Procedures.
* Code that will respond to an event that executes in response to the user.
* Example: Button Click
* A **Sub Procedure** is a block of code that will execute in response to a “call” from inside another block of code.
* When the Sub is called, its code statements are executed.
* Declare a Sub Procedure

**Private|Public Sub *subName* ()**

* + The Public|Private determines the accessibility of the procedure.
	+ Public is the default if it is not specified.
	+ Always concludes with a “End Sub” statement.
* Syntax

**Public Sub** *subName* **()**

 *code body here*

**End Sub**

Public or Private?

* Public Sub can be accessed from any module within the project
* Private Sub can be accessed only from within the same module
* Use Private unless you have a specific reason to declare something Public
* Calculate Sub Procedure

Private Sub Calculate()

Dim dblNum1, dblNum2, dblResult As Double

dblNum1 = Convert.ToDouble(txtNum1.Text)

dblNum2 = Convert.To Double(txtNum2.Text)

Select Case True

Case rdoAdd.Checked

dblResult = dblNum1 + dblNum2 …

. . .

End Sub

* “Calling” a Sub

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

 Call Calculate() ‘Calls the sub procedure

 ‘ “Call” is optional.

 Calculate() 'Calls the sub procedure

 End Sub

* The **( )** are required because values can be passed into the sub procedure.
* By Value or Reference
* In our previous example we did not pass any values (called arguments or actual parameters) into the procedure.
* These are the values or variables inside the () of the call.
* More complex procedures will usually have passed values.

Two ways - Value (Val) or Reference (Ref)

* Value Parameters
	+ Value Parameters are the default.
	+ Example:
	 Private Sub btnCalc\_Click(**ByVal** sender As
	 System.Object, **ByVal** e As System.EventArgs) Handles
	 btnCalc.Click
	+ Button actions are sub procedures too - this one is called when the button is clicked- and ByVal is used.
	+ We distinguish them by calling the **Event Procedures**.
	+ They will only execute if the event is called, such as a button click.
	+ **Value Parameters** pass the value to the sub procedure.
	+ Passing a **value** of a variable - not the variable itself.
	+ The value of the variable cannot be changed.
	+ So just the value is passed into the sub procedure- the actual variable remains separate with its own value.
	+ Use ByVal when no return value is needed.
	+ Think of ByVal as a “one way street”
	+ Only used as a local variable by the called procedure.
	+ After the procedure has executed, the value of the argument in the procedure call has not changed.
	+ The order of the arguments passed corresponds to the order of the parameters.
	+ The number of arguments in a procedure call must match the number of parameters in the in the procedure dbllaration.
	+ Arguments passed by value can be in the form of constants, variables, value or expressions.
	+ Variable arguments passed by value are not changed by the procedure.
	+ Value Parameter Example
	+ In a Button Click…

Dim dblNumber1, dblNumber2 As Double

dblNumber1 = Convert.To Double(txtNum1.Text)
dblNumber2 = Convert.To Double(txtNum2.Text)

Calculate (dblNumber1, dblNumber2)

Private Sub Calculate(ByVal dblNum1 As Double, ByVal dblNum2 As Double)

Dim dblResult As Double

Select Case True

Case rdoAdd.Checked

dblResult = dblNum1 + dblNum2
 lblResult.Text = dblResult.ToString

. . .

End Sub

* + Passing Arguments

Calculate(dblNumber1, dblNumber2)

* + - This calls the procedure and passes our 2 arguments - dblNumber1 and dblNumber2.

Private Sub Calculate(ByVal dblNum1 As Double, ByVal dblNum2 As Double)

* + - The second statement is the actual sub procedure with the same 2 arguments passed by value.
		- This also identifies the data types of the passed arguments.
		- In this case dblNum1 and dblNum2 are called **formal parameters**.
		- The number of arguments passed must equal the number identified in the call statement.
	+ Passing Arguments

Calculate(dblNumber1, dblNumber2)

Private Sub Calculate(ByVal dblNum1 As Double, ByVal dblNum2 As Double)

* The value is passed – The first actual argument from the call statement goes to the first formal argument/parameter.
* **IMPORTANT**: The Number, Data Type and Order of the arguments MUST match the number, data type and order of the parameters,
* Reference Parameters
	+ Does not create a new place in memory (storage)
	+ Represents the same storage location as the variable given in the argument “call”
	+ In Visual Basic, we pass reference parameters “By Ref”
	+ Passing ByRef passes a reference to the variable itself.
	+ The sub procedure CAN change the value of the variable.
	+ Think of ByRef as a “Two Way Street” in that the value comes back.
	+ An example of when to use ByRef would be with a procedure that calculates a value that is then returned to the main program.
	+ For example, the interest on a debt- the interest is returned to the main program where it is added to the debt to calculate a payment.
	+ Reference Parameter Sample
	+ In a Button Click…

Dim dblNumber1, dblNumber2, dblAnswer As Double

dblNumber1 = Convert.ToDouble(txtNum1.Text)
dblNumber2 = Convert. ToDouble(txtNum2.Text)

Calculate (dblNumber1, dblNumber2, dblAnswer)

lblResult.Text = dblAnswer.ToString

Private Sub Calculate(ByVal dblNum1 As Integer, ByVal dblNum2 As Integer, **ByRef** dblAns As Double)

Dim idblResult As Double

Select Case True

Case rdoAdd.Checked

dblAns = dblNum1 + dblNum2
 . . .

End Sub

Procedure Documentation

* Precondition
	+ Assumptions or initial requirements of a procedure.
* You should include preconditions above your procedure.
* Example:
PRE: x is an Integer and x > 0
* Postcondition
	+ A statement of what must be true at the end of the execution of a procedure if the procedure has worked properly.
* You should include preconditions above your procedure.
* Example:
POST: Result is calculated

Strong Typing

* In the previous examples, strong typing was used.
* This is assigning a data type to ALL variables including those in procedure calls.
* This is strongly recommended.
* According to MicroSoft . . .
	+ It enables IntelliSense support for your variables and parameters.
	+ This allows you to see their properties and other members as you type in your code.
	+ It allows the compiler to perform type checking.
	+ This helps catch statements that can fail at run time due to errors such as overflow.
	+ It also catches calls to methods on objects that do not support them.
	+ It results in faster execution of your code.
* One reason for this is that if you do not specify a data type for a programming element, the Visual Basic compiler assigns it the Object type.
* Your compiled code might have to convert back and forth between Object and other data types, which reduces performance.

Option Strict

* To enforce strong typing by default (the IDE will require all variables to have data types dbllared) turn on Option Strict.
* Type Option Strict On before all other code.
* In Tools🡪OptionsProjects and SolutionsVB Defaults Option Strict can be turned on by default for all programs.
* This also requires all conversions between data types be stated- not inferred. Examples given in these PowerPoint follow this rule using the Convert keyword or the ToString() method.

Functions

* Functions are a special type of procedure that return a value to the main program using a return statement.
* Functions must be given a data type when declared. The function ends with an “End Function” statement.
* For example:

Function ProcedureName(ByVal parameter1 As type, …) As ReturnType

 Statements

 Return value

End Function

* Functions are called from within a statement that will make use of the return value.
* The order of the arguments corresponds to the order of the parameters.
* Only ByVal parameters should be declared in a function because a function should not alter the arguments it has been passed.
* A function returns a single value and therefore must be used in a statement such as an assignment statement that makes use of the returned value.
* Sample Code

 dblResult = Calculate(dblNum1, dblNum2)
 'Calls a Function – returns a value

 Calculate(dblNum1, dblNum2)
 ‘Calls a sub procedure & does not return anything

* Notice the difference between this and a sub procedure.
	+ An assignment statement is used to assign the function (which returns a value) to a variable.
	+ A sub procedure does not return a value.
* Sample Code

Function Calculate(ByVal dblNum1 As Double, ByVal dblNum2 As Double) As Double

* The function call itself is given above. Remember all functions must be given a data type.
* Formal arguments/parameters are always ByVal.
* Last line of a function is always a statement with the Return keyword:
 Return dblResult
 'Return dblResult value to the main program
* Sample Code
	+ This is the code of the button:

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

Dim dblNum1, dblNum2, dblResult As Double

‘Validates input from the text boxes
 Try

 dblNum1 = Convert.ToDouble(txtNum1.Text)

 dblNum2 = Convert.ToDouble(txtNum2.Text)

 Catch ex As Exception
 MessageBox.Show(“Enter numeric values”)
 End Try

dblResult = Calculate(dblNum1, dblNum2) 'Calls the Function

‘Displays the result in the lblResult object.

lblResult.Text = dblResult.ToString()

End Sub

* Sample Code
* The code of the function is given in the notes for this slide. It is the same code as in the sub procedure example except for the Return statement.

Function Calculate(ByVal dblNum1 As Double, ByVal dblNum2 As Double) As Double

 Return dblResult 'Return dblResult to the main program

End Function

**Vocabulary**

* Modularization
* Sub Procedure
* Arguments
* Actual Parameter
* Event Procedure
* Value Parameter
* Formal Parameter
* Reference Parameter
* Precondition
* Postcondition
* Strong Typing
* Intellisense
* Function

**Code**

* Basic Sub Procedure
 Public Sub subName ()
 End Sub
* Sub Procedure with Value Parameters
 Public Sub subName (ByVal name As DataType, …)
 End Sub
* Sub Procedure with Reference Parameters
 Public Sub subName (ByRef name As DataType, …)
 End Sub
* Sub Procedure with Value & Reference Parameters
 Public Sub subName (ByVal name As DataType, ByRef name As
 DataType, …)
 End Sub
* Option Strict On
* Function

Function ProcedureName(ByVal parameter1 As type, …) As ReturnType

 Statements

 Return value/variable

End Function

Wrap it Up

* This PowerPoint described sub procedures and functions in Visual Basic.
* These allow you to break up and better segment code in complex applications.
* Remember the differences:
* Functions have a data type and always return a value, sub procedures do not.
* Sub procedures are called in code, functions are called in an assignment statement.