'-----------------------------------------------------------------------------------------------------------------------------------

'Last Update: Sept 25, 2013

'-----------------------------------------------------------------------------------------------------------------------------------

Option Explicit

Option Base 1

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.6. Optional Parameters To Procedures

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'This section describes two methods of implementing optional and a variable number of parameters to a VBA

'procedure.

'There are times when we may not be required to pass all the arguments in a parameter list to a procedure. This is typically the case

'when parameters later in the list are dependent on specific values of variables earlier in the list. To declare a parameter as optional,

'we include the Optional keyword in the parameter declaration. When we declare a parameter as optional, all subsequent parameters in the

'list must also be optional.

'The majority of procedures (Sub and Function procedures) use a fixed number of parameters and most often these parameters have explicit

'data types. This is all well and good, but there may be circumstances in which the number of parameters cannot be known until run-time.

'There are two methods that you can use to handle a variable number of parameters. The first, Optional Variants, defines a fixed number

'of parameters but makes these parameters optional, effectively putting an upper limit on the number of parameters than may be passed.

'The second method is to use a ParamArray type parameter. This allows for any number of parameters, including none. In either method,

'if your procedure requires some parameters but not others, declare all the required parameters first and then declare the optional

'parameters with either the Optional attribute or with a ParamArray type parameter after declaring the required parameters.

'i) Optional Variants

'You can declare one or more parameters as optional parameters of the Variant data type. With these optional Variant parameters, you

'can use the IsMissing function to determine whether a parameter was included or omitted. There are a few rules that govern the use

'optional parameters:

' The Optional keyword must be present to make a parameter optional.

' The data type should be (but need not be, see below) a Variant data type.

' The optional parameter(s) must be at the end of the parameter list.

' The IsMissing function will work only with parameters declared as Variant. It will return False when used with any other data type.

' User defined types (UTDs) cannot be optional parameters.

'We 'll examine each of the rules. First, the keyword Optional must be used in the parameter declaration. This keyword is what makes the

'parameter optional. Without it, the parameter is required. For example,

'Function Test\_Optional\_Parameter(L1 As Long, L2 As Long, Optional P1 As Variant, Optional P2 As Variant) As String

'In this example, L1 and L2 are required parameters and P1 and P2 are optional parameters. Since P1 and P2 are Variant types, we can use

'the IsMissing to determine whether the parameters were passed to the procedure.

'The data type should be a Variant type varible. Actually, this is only partially true. If you need to test specifically whether a parameter

'was actually passed to the procedure, you must declare it as a Variant so you can use the IsMissing function to determine if the parameter

'was passed. IsMissing may be used only with Variant types. It will return False for any other data type, even it that parameter is declared

'as Optional and is in fact missing. An optional parameter that is not a Variant will be assigned the default value for that data type (0

'for numeric data types, and empty string for String types, and Nothing for all object type variables. For example,

Sub Test\_Optional\_Parameter()

'P1 Is Missing. P2 Is Missing

Debug.Print Optional\_Parameter\_Func(1, 2)

'P1 Is Present (P1 = 3) P2 Is Missing

Debug.Print Optional\_Parameter\_Func(1, 2, 3)

'P1 Is Present (P1 = 3) P2 Is Present (P2 = 4)

Debug.Print Optional\_Parameter\_Func(1, 2, 3, 4)

End Sub

Function Optional\_Parameter\_Func(L1 As Long, L2 As Long, \_

Optional P1 As Variant, Optional P2 As Variant) As String

Dim S As String

If IsMissing(P1) = True Then

S = "P1 Is Missing."

Else

S = "P1 Is Present (P1 = " & CStr(P1) & ")" 'CStr convert a long value to a string (text)

End If

If IsMissing(P2) = True Then

S = S & " " & "P2 Is Missing"

Else

S = S & " " & "P2 Is Present (P2 = " & CStr(P2) & ")"

End If

Optional\_Parameter\_Func = S

End Function

'Here, both L1 and L2 are required but P1 and P2 are optional. Since both are Variant types, we can use IsMissing to determine whether the

'parameter was passed in. IsMissing returns True of the Variant parameter is omitted, or False is the Variant parameter is included. If the

'data type of the optional parameter is any data type other than Variant, IsMissing will return False.

'However, you can declare optional parameters of any data type and provide a default value to be used if that parameter is omitted. But the

'IsMissing function works only with Variant data types, so if you use an optional Long with a default value, there is no way to determine

'whether that parameter was actually passed to the procedure with a value that happens to be the same as the default value, or whether the

'parameter was omitted. For example,

'Function Optional\_Parameter\_Func2(Optional V As Variant, Optional L As Long = -1) As String

'Here, both V and L are optional, but only V is a Variant type, so only V may be tested with the IsMissing function. If L is omitted,

'IsMissing will return False since L is not a Variant. If omitted, L will have a value of -1. This may well be sufficient, but there is no

'way to determine whether L was actually passed into the procedure. If you write code such as this, you should choose a default value that

'will never occur if optional value is passed in. What that default value should be depends entirely on what the procedure does and the

'operating environment in which it is used.

'Optional parameters must be the last parameters declared for the procedure. That is, once one parameter is declared as Optional only other

'Optional parameters may follow it.

'The first procedure declarion shown below (GoodFunction) is valid because all the optional parameters are at the end of the

'parameter list. The second procedure declaration (BadFunction) is invalid because a required parameter (L2) follows an optional parameter

'(M1). You will get a compiler error ("Expected: Optional") if you attempt to use code like the BadFunction declaration.

'Function GoodFunction(L1 As Long, L2 As Long, Optional M1 As Variant, Optional M2 As Variant) As Variant

'Function BadFunction(L1 As Long, Optional M1 As Variant, L2 As Long, Optional M2 As Variant) As Variant

'A User Defined Type (UDT -- a structure declared with the Type keyword) cannot be an optional parameter, nor can a UDT be an element in a

'ParamArray parameter type. Code that attempts to do this will cause the compiler to emit an error.

'ii) ParamArray Parameter Type

'By using the Optional keyword, you can declare a function that accepts both required and optional parameters, but you are still limited to

'a fixed number of parameters. For example, the declaration: Sub Optional\_Parameters(L1 As Long, Optional P1 As Variant, Optional P2 As Variant)

'allows P1 and P2 as optional parameters, but this routine is still limited to a maximum of three parameters -- you can't pass in four or

'more parameters. Moreover, each optional parameter must be tested with IsMissing to determine whether it was passed. This can be onerous if

'you have a large number of optional parameters.

'The solution to this problem is the ParamArray parameter type. A ParamArray allows any number of parameters, including none at all, to be

'passed into a procedure (See http://www.tushar-mehta.com/publish\_train/xl\_vba\_cases/1005%20ParamArray.shtml). Note that a ParamArray allows

'for optional parameters following any number of required parameters. Only the ParamArray elements are option. Any parameters declared

'before the ParamArray are required. There are a few rules for using a ParamArray:

' a) The ParamArray variable must be an array of Variant data types.

' b) The ParamArray variable must be the last parameter in the parameter list. (The reason for this is that were another parameter to follow

' the ParamArray, it would be impossible for the code to determine where the ParamArray variables end and the subsequent variables begin.)

' c) No Optional parameters may appear in the parameter list. That is, the usages of Optional and ParamArray are mutually exclusive. You can

' have one or neither, but not both.

' d) The function must declare at most one ParamArray variable. It is illegal (and nonsensical if you really think about it) to have two

' ParamArray parameters declared for a procedure.

' e) Since a ParamArray parameter is an array, it is always passed by reference (ByRef) so any changes in the called procedure (which declares

' the ParamArray) to the elements in the ParamArray are changed in the calling procedure. (See Passing And Returning Arrays for more

' information about passing arrays.)

' f) The base of the ParamArray array is 0, regardless of the setting of the module's Option Base statement. If zero parameters are passed, the

' LBound is 0 and the UBound is -1. The LBound function returns the smallest subscript for the indicated dimension of an array. We will cover

' LBound for any dimension in Lesson 4.

' g) A User Defined Type (UDT) cannot be an element of a ParamArray parameter type.

'The following is an example of a procedure declaration that accepts one required parameter, L1, and then accepts any number of additional

'parameters in the ParamArray Args() parameter: Sub Test\_ParamArray(L1 As Long, ParamArray Args() As Variant)

'Since the ParamArray Args() parameter is an array, you can use normal array methods on the array. For example, you can loop through the

'stock prices (elements) of the m\_StockPrices (list):

Sub Test\_Print\_Mult\_Symbols()

Debug.Print Print\_Mult\_Symbols(4, "MSFT", "GOOG", "TD", "RY") 'Pass as a list

End Sub

Private Function Print\_Mult\_Symbols(ByVal NSIZE As Integer, ParamArray m\_StockSymbols() As Variant)

'ParamArray m\_StockSymbols() enables us to pass an arbitrary number of Stock Symbols into the function Print\_Mult\_Symbols. That is,

'ParamArray allows an indeterminate number of input arguments passed as either one-dimensional list or as an array of the type specified.

'Within the function definition, the parameter array m\_StockSymbols is treated as an array of its declared type, As Variant. To use a

'ParamArray, just specify the last parameter in a parameter list as a ParamArray.

Dim i As Variant

Dim j As Long

Dim TEXT\_STR As String

j = 0

For Each i In m\_StockSymbols

TEXT\_STR = TEXT\_STR & i & ", "

j = j + 1

'In this program a list of 4 Symbols is passed. In Lesson 4, we will take an in depth look at arrays. Also, notice the use of the

'For Each...Next loop structure as we discussed in the previous lesson.

Next i

If j <> NSIZE Then: NSIZE = j

TEXT\_STR = ("Porfolio of stocks contains " & NSIZE & " stocks. The Symbols are: ") & TEXT\_STR

Print\_Mult\_Symbols = TEXT\_STR

End Function

'Since the data type of the ParamArray m\_StockSymbols() As Variant declares Variant data type, the elements of m\_StockSymbols may contain

'anything, including numbers, strings, objects, and arrays, in addition to being Empty or NULL. Unless you have tight control over what is

'being passed in the array, your code should check the type of each Args element to ensure it is valid. For example, see the two

'versions of the SumUp function:

Sub Test\_SumUps()

Debug.Print SumUp1(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) 'Pass as a list

Debug.Print SumUp2(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) 'Pass as a list

End Sub

Function SumUp1(ParamArray Args() As Variant) As Double

Dim Sum As Double

Dim Ndx As Long

For Ndx = LBound(Args) To UBound(Args)

Sum = Sum + Args(Ndx)

Next Ndx

SumUp1 = Sum

End Function

Function SumUp2(ParamArray Args() As Variant) As Double

Dim Sum As Double

Dim Ndx As Long

For Ndx = LBound(Args) To UBound(Args)

'for IsNull, IsArray, IsObject, IsError see http://en.wikipedia.org/wiki/Is\_functions

If IsNull(Args(Ndx)) = False Then

If IsArray(Args(Ndx)) = False Then

If IsObject(Ndx) = False Then

If IsError(Ndx) = False Then

Sum = Sum + Args(Ndx)

End If

End If

End If

End If

Next Ndx

SumUp2 = Sum

End Function

'While all this testing may seem like overkill, you must remember that with a Variant, you have no control over what that variable might

'contain and your code should test for possible errors. You cannot have optional arguments within the ParamArray itself. If you omit an

'argument in a ParamArray an error is placed in that element of the array. For example,

'Call this function with code like the following:

Sub TEST\_FFF\_FUNC1()

Dim d As Double

d = FFF\_FUNC1(1, , 3, 4) 'Why at element 0?

Debug.Print d

End Sub

Function FFF\_FUNC1(ParamArray Args() As Variant) As Long

Dim N As Long

Dim d As Double

For N = LBound(Args) To UBound(Args)

If IsError(Args(N)) = True Then

Debug.Print "ERROR In ParamArray At Element: " & CStr(N)

Else

d = d + Args(N)

End If

Next N

FFF\_FUNC1 = d

End Function

'This will illustrate that the second value passed in the ParamArray (the missing element) is treated as an error by the code in function FFF.

'The array variable declared as a ParamArray has a LBound value of 0, regardless of the module's Option Base setting. If no parameters were

'passed via the ParamArray variable, the LBound of the ParamArray is 0 and the UBound of the ParamArray is -1. The number of elements in the

'ParamArray can be determine code similar to the following:

Sub TEST\_ARRAY\_SIZE\_FUNC()

Debug.Print NumArgs\_Func(1, 2, 3, 4)

End Sub

Function NumArgs\_Func(ParamArray Args() As Variant) As Long

Dim NumArgs As Long

NumArgs = UBound(Args) - LBound(Args) + 1

'Debug.Print LBound(Args), UBound(Args)

NumArgs\_Func = NumArgs

End Function