How to Use Excel VBA Variable Data Types

You will learn to create the different types of Excel VBA (Visual Basic for Applications) Variable Data Types. VBA makes life easy for programmers because it can handle all the details involved in dealing with data automatically. For...

Part 1 of 4:

Variable Data Types

1.

Data Type	Bytes Used	Range of Values
Byte	1 byte	0 to 255
Boolean	2 bytes	True or False
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,647
Single	4 bytes	-3.402823E38 to -1.401298E-45 (for negative values); 1.401298E-45 to 3.402823E38 (for positive values)
Double	8 bytes	-1.79769313486232E308 to -4.94065645841247E-324 (negative values); 4.94065645841247E-324 to 1.79769313486232E308 (for positive values)
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	12 bytes	+/-79,228,162,514,264,337,593,543,950,335 with no decimal point; +/-7.9228162514264337593543950335 with 28 places to the right of the decimal
Date	8 bytes	January 1, 0100 to December 31, 9999
Object	4 bytes	Any object reference
String (variable length)	10 bytes + string length	0 to approximately 2 billion characters
String (fixed length)	Length of string	1 to approximately 65,400 characters
Variant (with numbers)	16 bytes	Any numeric value up to the range of a double data type. It can also hold special values, such as Empty, Error, Nothing, and Null.
Variant (with characters)	22 bytes + string length	0 to approximately 2 billion
User-defined	Varies	Varies by element

Learn from the above table the Excel VBA Variable Data Types.

2. **Learn that generally, it's best to use the data type that uses the smallest number of bytes yet still can handle all the data that will be assigned to it.** When VBA works with data, execution speed is partially a function of the number of bytes that VBA has at its disposal. Thus, the fewer bytes used by data, the faster VBA can access and manipulate the data.
    1. The Decimal data type is rather unusual because you can't actually declare it. It is a subtype of a variant. You need to use the VBA CDec function to convert a variant to the Decimal data type.
3. **Learn various Assignment Statements.** The following list contains some examples of assignment expressions that use various variable types. The variable names are to the left of the equals sign. Each statement assigns the value to the right of the equals sign to the variable on the left.

1. 1. DataEntered = True
   2. DateStart = #04/18/2015#
   3. Interest_Rate = 0.025
   4. LoanAmount = 2000000.00
   5. MyNumber = TheNumber * 1.25
   6. UserName = 'Terrence O'Malley'
   7. x = 10
   8. x = x + 1
4. **Learn that VBA has many reserved words, which are words that you can't use for variable or procedure names.** If you attempt to use one of these words, you get an error message. For example, although the reserved word Next or True might make a very descriptive variable name, the following instructions generate a syntax error: Next = 64; True = True.
5. **Learn that syntax error messages aren't always descriptive.** If an instruction produces a strange error message, check the VBA Help system to ensure that your variable name doesn't have a special use in VBA. If the Auto Syntax Check option is turned on you get the error: *Compile error: Expected: variable*. If Auto Syntax Check is turned off, attempting to execute this statement results in: *Compile error: Syntax error*. It would be more helpful if the error message were something like *Reserved word was used as a variable*.
6. **Learn that for worksheet calculation, Excel uses the Double data type, so that's a good choice for processing numbers in VBA when you don't want to lose any precision.** For integer calculations, you can use the Integer type which is limited to values less than or equal to 32,767. Otherwise, use the Long data type. Using the Long data type even for values less than 32,767 is recommended, because this data type may be a bit faster than using the Integer type. When dealing with Excel worksheet row numbers, you want to use the Long data type because the number of rows in a worksheet exceeds the maximum value for the Integer data type.

Part 2 of 4:

Declaring Variables

1. **If you don't declare the data type for a variable that you use in a VBA routine, VBA uses the default data type, Variant.** Data stored as a Variant changes type, depending on what you do with it.
   1. The following procedure demonstrates how a variable can assume different data types:
      1. Sub VariantDemonstration()
      2. TheVar = '124'
      3. TheVar = MyVar / 2
      4. TheVar = 'Answer: ' & TheVar
      5. MsgBox TheVar
      6. End Sub
      7. In the VariantDemonstration procedure, TheVar starts out as a three-character string. Then this string is divided by two and becomes a numeric data type. Next, TheVar is appended to a string, converting TheVar back to a string. The MsgBox statement displays the final string: Answer: 62.
   2. To further demonstrate the potential problems in dealing with Variant data types, try executing this procedure:
      1. Sub VariantDemonstration2()
      2. TheVar = '124'
      3. TheVar = TheVar + TheVar

4. TheVar = 'Answer: ' & TheVar
5. MsgBox TheVar
6. End Sub
7. The message box displays Answer: 124124. This is probably not what you wanted. When dealing with variants that contain text strings, the + operator performs string concatenation.

2. **You can use the VBA TypeName function to determine the data type of a variable.** Here's a modified version of the previous procedure. This version displays the data type of TheVar at each step.
    1. You see that it starts out as a string, is then converted to a double, and finally ends up as a string again.
        1. Sub VariantDemonstration3()
        2. TheVar = '124'
        3. MsgBox TypeName(TheVar)
        4. TheVar = TheVar / 2
        5. MsgBox TypeName(TheVar)
        6. TheVar = 'Answer: ' & TheVar
        7. MsgBox TypeName(TheVar)
        8. MsgBox TheVar
        9. End Sub

Part 3 of 4:

User-Defined Data Types

1. **Learn that VBA lets you create custom, or user-defined, data types.** A user-defined data type can ease your work with some types of data.
    1. For example, if your application deals with customer information, you may want to create a user-defined data type named *CustomerData*:
        1. Type CustomerData
        2. Company As String
        3. Contact As String
        4. RegionCode As Long
        5. Sales As Double
        6. End Type
2. **You define custom data types at the top of your module, before any procedures.**
3. **After you create a user-defined data type, you use a Dim statement to declare a variable as that type.** Usually, you define an array. For example:
    1. Dim Customers(1 To 100) As CustomerData
    2. Each of the 100 elements in this array consists of four components (as specified by the user-defined data type, CustomerData).
    3. You can refer to a particular component of the record as follows:
        1. Customers(1).Company = 'Ace Tools'
        2. Customers(1).Contact = 'Tim Roberts'
        3. Customers(1).RegionCode = 1
        4. Customers(1).Sales = 150000.00
    4. You can also work with an element in the array as a whole. For example, to copy the information from Customers(1) to Customers(2), use this instruction:
        1. Customers(2) = Customers(1)
    5. The preceding example is equivalent to the following instruction block:

1. Customers(2).Company = Customers(1).Company
2. Customers(2).Contact = Customers(1).Contact
3. Customers(2).RegionCode = Customers(1).RegionCode
4. Customers(2).Sales = Customers(1).Sales

Part 4 of 4:

Arrays

1. **An array is a group of elements of the same type that have a common name.** You refer to a specific element in the array by using the array name and an index number. For example, you can define an array of 12 string variables so that each variable corresponds to the name of a month. If you name the array MonthNaming, you can refer to the first element of the array as MonthNaming(0), the second element as MonthNaming(1), and so on, up to MonthNaming(11).
2. **You declare an array with a Dim or Public statement, just as you declare a regular variable.** You can also specify the number of elements in the array. You do so by specifying the first index number, the keyword To, and the last index number — all inside parentheses.
    1. For example, here's how to declare an array comprising exactly 100 integers:
        1. Dim TheArray(1 To 100) As Integer
3. **When you declare an array, you need specify only the upper index, in which case VBA assumes that 0 is the lower index.**
    1. Therefore, the two statements that follow have the same effect:
        1. Dim TheArray(0 to 100) As Integer
        2. Dim TheArray(100) As Integer
        3. In both cases, the array consists of 101 elements.
4. **By default, VBA assumes zero-based arrays.** If you would like VBA to assume that 1 is the lower index for all arrays that declare only the upper index, include the following statement before any procedures in your module: Option Base 1
5. **When you want to declare multidimensional arrays, do the following:**
    1. The array examples in the preceding section are one-dimensional arrays. VBA arrays can have up to 60 dimensions, although you'll rarely need more than three dimensions (a 3-D array).
    2. The following statement declares a 100-integer array with two dimensions (2-D):
        1. Dim TheArray(1 To 10, 1 To 10) As Integer
        2. You can think of the preceding array as occupying a 10-x-10 matrix. To refer to a specific element in a 2-D array, you need to specify two index numbers. For example, here's how you can assign a value to an element in the preceding array:
        3. TheArray(3, 4) = 125
6. **Following is a declaration for a 3-D array that contains 1,000 elements (visualize this array as a cube):**
    1. Dim TheArray(1 To 10, 1 To 10, 1 To 10) As Integer
        1. Reference an item within the array by supplying three index numbers:
        2. TheArray(4, 8, 2) = 0
7. **When you want to declare a dynamic array, use a blank set of parentheses.** A dynamic array doesn't have a preset number of elements.
    1. Dim TheArray() As Integer
    2. Before you can use a dynamic array in your code, however, you must use the ReDim statement to tell VBA how many elements are in the array. You can use a variable to assign the number of elements in an array.

3. Often the value of the variable isn't known until the procedure is executing. For example, if the variable x contains a number, you can define the array's size by using this statement:
    1. ReDim TheArray (1 to x)
4. You can use the ReDim statement any number of times, changing the array's size as often as you need to. When you change an array's dimensions the existing values are destroyed. If you want to preserve the existing values, use ReDim Preserve. For example:
    1. ReDim Preserve TheArray (1 to z)