An array is a consecutive group of memory locations that all have the same name and the same type. To refer to a particular location or element in the array, we specify the array name and the array element position number.

The Individual elements of an array are identified using an index. Arrays have upper and lower bounds and the elements have to lie within those bounds. Each index number in an array is allocated individual memory space and therefore users must evade declaring arrays of larger size than required. We can declare an array of any of the basic data types including variant, [user-defined types](http://visualbasic.freetutes.com/learn-vb6/lesson6.1.html) and object variables. The individual elements of an array are all of the same data type.

**Declaring arrays**

Arrays occupy space in memory. The programmer specifies the array type and the number of elements required by the array so that the compiler may reserve the appropriate amount of memory. Arrays may be declared as Public (in a code module), module or local. Module arrays are declared in the general declarations using keyword Dim or Private. Local arrays are declared in a procedure using Dim or Static. Array must be declared explicitly with keyword "As".

There are two types of arrays in Visual Basic namely:

http://visualbasic.freetutes.com/learn-vb6/images/dot.JPG**Fixed-size array :** The size of array always remains the same-size doesn't change during the program execution.

http://visualbasic.freetutes.com/learn-vb6/images/dot.JPG**Dynamic array :** The size of the array can be changed at the run time- size changes during the program execution.

**Fixed-sized Arrays**

When an upper bound is specified in the declaration, a Fixed-array is created. The upper limit should always be within the range of long data type.

Declaring a fixed-array

Dim numbers(5) As Integer

In the above illustration, numbers is the name of the array, and the number 6 included in the parentheses is the upper limit of the array. The above declaration creates an array with 6 elements, with index numbers running from 0 to 5.

If we want to specify the lower limit, then the parentheses should include both the lower and upper limit along with the To keyword. An example for this is given below.

Dim numbers (1 To 6 ) As Integer

In the above statement, an array of 10 elements is declared but with indexes running from 1 to 6.

A public array can be declared using the keyword Public instead of Dim as shown below.

Public numbers(5) As Integer

**Multidimensional Arrays**

Arrays can have multiple dimensions. A common use of multidimensional arrays is to represent tables of values consisting of information arranged in rows and columns. To identify a particular table element, we must specify two indexes: The first (by convention) identifies the element's row and the second (by convention) identifies the element's column.

Tables or arrays that require two indexes to identify a particular element are called two dimensional arrays. Note that multidimensional arrays can have more than two dimensions. Visual Basic supports at least 60 array dimensions, but most people will need to use more than two or three dimensional-arrays.

The following statement declares a two-dimensional array 50 by 50 array within a procedure.

Dim AvgMarks ( 50, 50)

It is also possible to define the lower limits for one or both the dimensions as for fixed size arrays. An example for this is given here.

Dim Marks ( 101 To 200, 1 To 100)

An example for three dimensional-array with defined lower limits is given below.

Dim Details( 101 To 200, 1 To 100, 1 To 100)

**Static and dynamic arrays**

Basically, you can create either static or dynamic arrays. Static arrays must include a fixed number of items, and this number must be known at compile time so that the compiler can set aside the necessary amount of memory. You create a static array using a Dim statement with a constant argument:

' This is a static array.   
Dim Names(100) As String

Visual Basic starts indexing the array with 0. Therefore, the preceding array actually holds 101 items.

Most programs don't use static arrays because programmers rarely know at compile time how many items you need and also because static arrays can't be resized during execution. Both these issues are solved by dynamic arrays. You declare and create dynamic arrays in two distinct steps. In general, you declare the array to account for its visibility (for example, at the beginning of a module if you want to make it visible by all the procedures of the module) using a Dim command with an empty pair of brackets. Then you create the array when you actually need it, using a ReDim statement:

' An array defined in a BAS module (with Private scope)  
Dim Customers() As String  
...  
Sub Main()  
' Here you create the array.  
ReDim Customer(1000) As String  
End Sub

If you're creating an array that's local to a procedure, you can do everything with a single ReDim statement:

Sub PrintReport()  
' This array is visible only to the procedure.  
ReDim Customers(1000) As String  
' ...  
End Sub

If you don't specify the lower index of an array, Visual Basic assumes it to be 0, unless an Option Base 1 statement is placed at the beginning of the module. My suggestion is this: Never use an Option Base statement because it makes code reuse more difficult. (You can't cut and paste routines without worrying about the current Option Base.) If you want to explicitly use a lower index different from 0, use this syntax instead:

ReDim Customers(1 To 1000) As String

Dynamic arrays can be re-created at will, each time with a different number of items. When you re-create a dynamic array, its contents are reset to 0 (or to an empty string) and you lose the data it contains. If you want to resize an array without losing its contents, use the ReDim Preserve command:

ReDim Preserve Customers(2000) As String

When you're resizing an array, you can't change the number of its dimensions nor the type of the values it contains. Moreover, when you're using ReDim Preserve on a multidimensional array, you can resize only its last dimension:

ReDim Cells(1 To 100, 10) As Integer  
...  
ReDim Preserve Cells(1 To 100, 20) As Integer ' This works.  
ReDim Preserve Cells(1 To 200, 20) As Integer ' This doesn't.

Finally, you can destroy an array using the Erase statement. If the array is dynamic, Visual Basic releases the memory allocated for its elements (and you can't read or write them any longer); if the array is static, its elements are set to 0 or to empty strings.

You can use the LBound and UBound functions to retrieve the lower and upper indices. If the array has two or more dimensions, you need to pass a second argument to these functions to specify the dimension you need:

Print LBound(Cells, 1) ' Displays 1, lower index of 1st dimension  
Print LBound(Cells) ' Same as above  
Print UBound(Cells, 2) ' Displays 20, upper index of 2nd dimension  
' Evaluate total number of elements.  
NumEls = (UBound(Cells) \_ LBound(Cells) + 1) \* \_  
(UBound(Cells, 2) \_ LBound(Cells, 2) + 1)

**Arrays within UDTs**

UDT structures can include both static and dynamic arrays. Here's a sample structure that contains both types:

Type MyUDT  
StaticArr(100) As Long  
DynamicArr() As Long  
End Type  
...  
Dim udt As MyUDT  
' You must DIMension the dynamic array before using it.  
ReDim udt.DynamicArr(100) As Long  
' You don't have to do that with static arrays.  
udt.StaticArr(1) = 1234

The memory needed by a static array is allocated within the UDT structure; for example, the StaticArr array in the preceding code snippet takes exactly 400 bytes. Conversely, a dynamic array in a UDT takes only 4 bytes, which form a pointer to the memory area where the actual data is stored. Dynamic arrays are advantageous when each individual UDT variable might host a different number of array items. As with all dynamic arrays, if you don't dimension a dynamic array within a UDT before accessing its items, you get an error 9—"**Subscript out of range**."

**More Topics on Visual Basic 6 Arrays**

* [Visual Basic 6 Arrays and variants](http://visualbasic.freetutes.com/learn-vb6/vb6-arrays-variants.html)
* [Assigning and returning arrays](http://visualbasic.freetutes.com/learn-vb6/assigning-and-returning-arrays.html)
* [Byte arrays](http://visualbasic.freetutes.com/learn-vb6/byte-arrays.html)
* [Inserting and deleting items using arrays](http://visualbasic.freetutes.com/learn-vb6/inserting-deleting-using-arrays.html)
* [Sorting using Arrays](http://visualbasic.freetutes.com/learn-vb6/sorting-using-arrays.html)
* [Arrays of arrays](http://visualbasic.freetutes.com/learn-vb6/arrays-of-arrays.html)