Arrays in VBA

Objectives: At the end of this lesson you shall be able to
- **describe and declare an array in VBA**
- **differentiate between static and dynamic arrays**
- **declare, populate and read a multidimensional array**
- **describe the redim and preserve statements in VBA.**

Introduction

An Array is a group of variables of the same data type and with the same name. If we have a list of items which are of similar type to deal with, we need to declare an array of variables instead of using a variable for each item. For example, if we need to enter ten names, instead of declaring ten different variables for each name, we need to declare only one array holding all the names. The individual element or item in the array is identified by its index or subscript.

When arrays are used, data is stored in an organized way. Apart from this working with the data is easy and faster when iterations are done using the Loop statements like For… Next etc. on the Arrays. The following example declares an array variable to hold ten students in a school.

Dim students(10) As Integer

The array "students" in the preceding example contains ten elements. The indices of the elements range from 0 through 9 by default. The variables in the Array are now identified as students(0), students(1) etc.indicating the first element and second element etc.respectively.

Types of Arrays :

1 Static Arrays

2 Dynamic Arrays

Static array

A static array is an array that is sized in the Dim statement that declares the array. E.g.,

Dim Students(10) as String

Dim StaticArray(1 To 10) As Long

You cannot change the size or data type of a static array. When you erase a static array, no memory is freed. Erase simply sets all the elements to their default value (0, vbNullString, Empty, or Nothing, depending on the data type of the array).

Declaring an Array

You declare an array variable using the Dim statement.

Dim StudentName(3) As String

Arrays are also declared in another method where the type or the variable name with one or more pairs of parentheses is added to indicate that it will hold an array. After you declare the array, you can define its size by using the ReDim Statement.

The following example declares a one-dimensional array variable and also specifies the dimensions of the array by using the ReDim Statement.

Dim arr As Integer()

ReDim arr(10)

The following example declares a multidimensional array variable by placing commas inside the parentheses to separate the dimensions.

Dim arrayName (num1, num2) as datatype

To declare a jagged array variable, add a pair of parentheses after the variable name for each level of nested array.

Dim arr()()() As integer

In VBA arrays you can specify any value for the lower and upper bounds of the array. Element 0 need not be the first element in the array. For example, the following is perfectly legal code (as long as the lower bound is less than or equal to the upper bound -- an error is generated if the lower bound is greater the upper bound).

If you don't explicitly declare the lower bound of an array, the lower bound will be assumed to be either 0 or 1, depending on value of the Option Base statement, if present. If Option Base is not present in the module, 0 is assumed. For example, the code

Dim Arr(10) As Long

declares an array of either 10 or 11 elements. The declaration does not specify the number of elements in the array. Instead, it specifies the upper bound of the array. If your module does not contain an "Option Base" statement, the lower bound is assumed to be zero, and the declaration above is the same as : Dim Arr(0 To 10) As Long

Copyright Free under CC BY Licence

If you have an Option Base statement of 0 or 1, the lower bound of the array is set to that value.

Thus, the code : Dim Arr(10) As Long is the equivalent of either Dim Arr(0 To 10) As Long or Dim Arr(1 To 10) As Long, depending on the value of the Option Base.

It is a good programming practice to specify both the lower and upper bounds of the array to avoid bugs when copying and pasting code between modules or elsewhere.

Storing values in an array

Arrays can be populated in the following ways

1 Marks(0)=55
  Marks(1)=67
  Marks(2)=55
  Marks(3)=67
  Marks(4)=74
2 Dim marks As Integer() = {55, 67, 87, 48, 90, 74}
3 Dim marks = New Integer() {1, 2, 4, 8}
4 Dim doubles = {1.5, 2, 9.9, 18}

You can explicitly specify the type of the elements in an array that's created by using an array literal. In this case, the values in the array literal must widen to the type of the elements of the array. The following code example creates an array of type Double from a list of integers: Dim marks As Double() = {55, 67, 87, 48, 90, 74}

Iterating through an Array

Loop statements like for… next, Do …while etc. can be used with arrays to retrieve their values.An example of such a code is shown below.

```
Sub array_test()
Dim arr(5) As Integer
Dim n As Integer
arr(0) = 89
arr(1) = 56
arr(2) = 78
arr(3) = 45
arr(4) = 99
For n = LBound(arr) To UBound(arr) - 1
Debug.Print arr(n)
Next
End Sub
```

Here LBound() and UBound() functions return the Lower and Upper Bounds of the Array.

Multi dimensional Arrays

Multi dimensional arrays have more than one row or one column.

For ex.Dim MyArray(5, 4) As Integer

Dim MyArray(1 To 5, 1 To 6) As Integer

In the following ex. we will define an array with 3 elements each in two rows

```
Sub array_test()
Sub array_test()
Dim m, n As Integer
Dim arr(2, 4) As String
arr(0, 0) = "printer"
arr(0, 1) = "scanner"
arr(0, 2) = "mouse"
arr(0, 3) = "monitor"
arr(1, 0) = "usb"
arr(1, 1) = "ps2"
arr(1, 2) = "firewire"
arr(1, 3) = "serial"
For m = 0 To 2
   For n = 0 To 3
  Debug.Print arr(m, n); Spc(2);
   Next n
  Debug.Print
Next m
End Sub
```

Dynamic Arrays

A dynamic array is an array that is not sized in the Dim statement. Instead, it is sized with the ReDim statement. Dynamic array variables are useful when we don't know in advance how many elements need to be stored in the array or when we need to change the array dimensions at a later stage.

Copyright Free under CC BY Licence

E.g. : Dim DynamicArray() As Long

ReDim DynamicArray(1 To 10)

If an array is sized with the ReDim statement, the array is said to be allocated(either static array or a dynamic array).Static arrays are always allocated and never empty.You can change the size of a dynamic array, but not the data type. When you Erase a dynamic array, the memory allocated to the array is released. You must ReDim the array in order to use it after it has been Erased.

If a dynamic array has not yet been sized with the ReDim statement, or has been deallocated with the Erase statement, the array is said to be empty or unallocated. Static arrays are never unallocated or empty.

ReDim Statement:

You may declare a dynamic variable with empty parentheses ie. leave the index dimensions blank. You can thereafter size or resize the dynamic array that has already been declared, by using the ReDim statement. To resize an array, it is necessary to provide the upper bound, while the lower bound is optional. If you do not mention the lower bound, it is determined by the Option Base setting for the module, which by default is 0. You can specify Option Base 1 in the Declarations section of the module and then index will start from 1. This will mean that the respective index values of an array with 3 elements will be 1, 2 and 3. Not entering Option Base 1 will mean index values of 0, 1 and 2.

The following example declares an array called A1 as a dynamic array. The array's size is not set and then it is resized to 3 elements (by specifying Option Base 1)

```
Sub arr_test()

'declare a dynamic array

Dim A() As String

ReDim A(3) As String

A(1) = "COPA"

A(2) = "DTPO"

A(3) = "MASE"

debug.print  A(1)  & " , " & A(2) & " , " & A(3)

End Sub
```

When you use the ReDim keyword, you erase any existing data currently stored in the array.

For ex. add another element to the array mentioned in the example above using the redim statement as follows and assign a value to it.

```
ReDim A(4) As String

A(4) = "CHNM"
```

Now when the array values are displayed again, the earlier values will all be blank, since they are erased by the redim statement. The example below shows this:

```
Sub arr_test()

'declare a dynamic array

Dim A() As String

ReDim A(3) As String

A(1) = "COPA"

A(2) = "DTPO"

A(3) = "MASE"

ReDim A(4) As String

A(4) = "CHNM"

Debug.Print A(1) & " , " & A(2) & " , " & A(3) & "," & A(4)

End Sub
```

The result of this program will be , , ,CHNM

To resize the array without losing the existing data, you should use " Preserve " along with Redim. For ex. ReDim Preserve A(4) As String.

Copyright Free under CC BY Licence

String manipulation in VBA

Objectives: At the end of this lesson you shall be able to
- **describe the string concatenation functions in VBA**
- **describe the string conversion functions in VBA**
- **describe the string extraction functions in VBA**
- **describe the string formatting functions in VBA.**

Introduction

A string, in VBA, is a type of data variable which can consist of text, numerical values, date and time and alphanumeric characters. Strings are a frequently used to store all kinds of data and are important part of VBA programs. To declare a variable for it, you can use either String or the Variant data types.

String Manipulation

VBA has a robust set of functions for string handling. The following are some examples of where you might use string functions:

- Checking to see whether a string is contained another string
- Parsing out a portion of a string
- Replacing parts of a string with another value
- Finding the length of the string etc.

String Concatenation

String concatenation or joining two or more strings can be done by the "+" Operator

Example : Dim A, B, C As String

A = "www"

B = " and"

C = " the Internet"

Debug.Print A + B + C

This will print "www and the Internet"

If it is required to join two or more different data types variable, then the "&" operator can be used.

Example : Dim person As String, pay As Integer

person = "Jaya"

pay = 25000

Debug.Print "The payment for "& person & " is " & pay

This will print: The payment for Jaya is 25000

Len() function

Returns an integer containing either the number of characters in a string or the nominal number of bytes required to store a variable.

Syntax : Len (String)

Example : 1

Dim str As String

str = "Computer operator and programming assistant"

Debug.Print "The length of the string is "& Len(str)

This will print : The length of the string is 43

Example : 2

Dim p, q As Integer, r As Double, dob As Date

p = Sqr(25)

q = 3333

r = 45.6789

dob = #1/1/1990#

Debug.Print "Size of p Is : " & Len(p)

Debug.Print "Size of q Is : " & Len(q)

Debug.Print "Size of r Is : " & Len(r)

Debug.Print "Size of dob Is : " & Len(dob)

Thiswill print : Size of p Is : 1

Size of q Is : 2

Size of r Is : 8

Size of dob Is : 8

Copyright Free under CC BY Licence

Left()

Returns a string containing a specified number of characters from the left side of a string.

Syntax: Left(String ,Int)

Right()

Returns a string containing a specified number of characters from the right side of a string.

Syntax: Right(String, Int)

Mid()

Returns a string that contains characters from a specified string.

Syntax:Mid(String, Int, Int)

Returns a string that contains all the characters starting from a specified position in a string

Syntax: Mid(String, Int, Int)

Returns a string that contains a specified number of characters starting from a specified position in a string.Examples are :

1 Dim s AsString
s =" Indiana Jones"
debug.print Left(s)
This will print : India

2 Dim s AsString
s =" FUNDAMENTALLY"
debug.print right(s, 5)
This will print : TALLY

3 Dim s AsString
s =" wholehearted"
debug.print mid(s, 6)
This will print : hearted

4 Dim s AsString
s =" wholehearted"
debug.print mid(s, 6,4)
This will print : hear

Ltrim()

Returns a string containing a copy of a specified string with no leading spaces (LTrim)

Syntax :LTrim(String)

RTrim()

Returns a string containing a copy of a specified string with no trailing spaces.

Syntax :RTrim(String)

Trim()

Returns a string containing a copy of a specified string with no leading or trailing spaces.

Syntax : Trim(String)

Examples: Dim A as String

A = " Adjustment "

Debug.Print "For everyone" &LTrim(A) & "is a must"

Debug.Print "For everyone"; RTrim(A) & "is a must"

Debug.Print "For everyone"; Trim(A) & "is a must"

This will print : For everyoneAdjustment is a must

For everyone Adjustmentis a must

For everyone Adjustmentis a must

Instr()

Returns an integer specifying the start position of the first occurrence of one string within another.

Syntax: InStr([start,]string1, string2[, compare])

Example:

A = "hairdresser"

B = "dress"

Debug.Print "The second string starts at position no. " &InStr(1, A, B) "

This will print : The second string starts at position no. 5

Replacing Strings

The Replace() function replaces a sequence of characters in a string with another set of characters.

Replace(source_string, find_string, replacement_string).

Eample: Debug.Print Replace("majordrawback", "drawback", "advantage")

Debug.Print Replace("majority", "aj", "in", 1)

Debug.Print Replace("think and think", "i", "a", 1, 1)

This will print : major advantage

minority

thank and think

Copyright Free under CC BY Licence

Val()

The VAL() function accepts a string as input and returns the numbers found in that string.The VAL function will stop reading the string once it encounters the first non-numeric character. This does not include spaces.

Syntax: Val(String)

Example: Dim s1, s2, s3 As String

s1 = "6 feet 1 inch is his height"

s2 = "5 - 6 kms is the distance to my office from here"

s3 = "011 22222222 is my telephone number"

Debug.Print Val(s1)

Debug.Print Val(s2)

Debug.Print Val(s3)

This will print : 6

5

1122222222

The String Conversion Functions

LCase()

Returns a string or character converted to lowercase.

Syntax :LCase(String)

UCase()

Returns a string or character converted to uppercase.

Syntax :UCase(String)

Example:

Dim A, B as String

A="IF YOU FEAR YOU WILL BECOME WEAK"

B = "be a strong person"

Debug.PrintLcase(A)

Debug.PrintUCase(B)

This will print: if you fear you will become weak

BE A STRONG PERSON

Str()

The Str() function converts a number to a string.

CStr()

The CStr() function is used to convert any type of value to a string.

Syntax: Str(number as variant)

Example:

```
1  Dim Number As Double
   Number = 1450.5
   Debug.Print "The string is "&str(Number)
   This will print : The string is 1450.5
2  Dim Date_of_birthAs Date
   Date_of_birth = #1/1/1990#
   Debug.Print CStr(Date_of_birth)
   This will print : 01/01/1990
```

Asc()

The Asc() function returns an Integer value representing the ASCII code corresponding to a character or the first character in a string

Syntax :Asc(String)

Example :Asc("A") will return 65

Chr()

The Chr() Function returns the character associated with the specified ASCII code.

Syntax :Chr(Integer)

Example :Chr(68) will return the character "D"

Reversing a String

StrReverse(String)

StrReverse() returns a string in which the character order of a specified string is reversed.

Example: Dim A As String

A = "desserts"

Debug.Print StrReverse(A)

This will print : stressed

Format() function

The format() function returns a Variant (String) containing an expression formatted according to instructions contained in a format expression. It can be used to return formatted dates as well as formatted strings.

Copyright Free under CC BY Licence

Syntax(for FormattingStrings) : Format(String, Format)

User-Defined String Formats (Format Function)

You can use any of the following characters to create a format expression for strings:

Example :Dim x As String

Character	Description
@	Character placeholder. Display a character or a space. If the string has a character in the position where the at symbol (@) appears in the format string, display it; otherwise, display a space in that position. Placeholders are filled from right to left unless there is an exclamation point character (!) in the format string.
&	Character placeholder. Display a character or nothing. If the string has a character in the position where the ampersand (&) appears, display it; otherwise, display nothing. Placeholders are filled from right to left unless there is an exclamation point character (!) in the format string.
<	Force lowercase. Display all characters in lowercase format.
>	Force uppercase. Display all characters in uppercase format.
!	Force left to right fill of placeholders. The default is to fill placeholders from right to left.

x = "change case"

Debug.Print Format(x, ">")

This will print "CHANGE CASE"

Copyright Free under CC BY Licence