Looping statements in VBA

Objectives: At the end of this lesson you shall be able to
- **describe the "for" loops in VBA**
- **describe the "do" loops in VBA**
- **explain the use of the "exit" statement in VBA loops**
- **write appropriate code to perform repetitive tasks.**

Introduction

There may be many situations where you need to perform a task repeatedly / a certain number of times. In such cases the code for the task is placed inside a loop and the program iterates or repeats through the loop a certain number of times ie. till a certain condition is met. Some examples of such repetitive tasks are:

a Printing a text or number n number of times.

b Generating a sequence or series of numbers.

c Generating a table of certain calculations.

d Searching / Re arranging a set of numbers etc.

VBA provides the following types of loops to handle looping requirements (Refer Table 1)

Table 1

Loop Type	Description
for next loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do....until loop	Repeats a statement or group of statements until a condition is met.
do….while loop	Repeats a statement or group of statements as long as the condition is true.

The For Loop

The For ... next loop sets a variable to a specified set of values, and for each value, runs the VBA code inside the loop. For Ex.

```
For n = 1 To 10

debug.print  n

Next n
```

In this example, the initial value of n is set to 1, and the loop code, ie. printing the value of n is performed.The value of n is set to the next value which is by default an increment of 1. Thus this loop is executed 10 times and would print the numbers 1 to 10. The for statement in the above code is the same as For n = 1 To 10 Step 1 since the default increment is 1

The same code will print numbers from 10 to 1 if the step is changed to a negative value as shown below.

```
For n = 10 To 1 Step -1

debug. print  n

Next n
```

Similarly, the following Ex. would add all the numbers from 1 to 10 and print the sum.

```
Dim n, sum as integer

Sum=0

For n = 1 To 10

sum=sum + n

debug. print  sum

Next n
```

The For Each Loop

The For Each loop is similar to the For ... Next loop but, instead of looping through a set of values for a variable, it loops through every object within a set of objects. The following example would print the names of all the worksheets.

```
Dim ws As Worksheet

For each ws in Worksheets

debug. print ws.name

Next ws
```

The Exit For Statement

If you need to end the For loop before the end condition is reached or met, simply use the END FOR in conjunction with the IF statement. In the example given below, we exit the for loop prematurely and before the end condition is

met. The for example given below, the loop exits when n reaches a value of 5.

```
For n = 0 To 10

debug.print n

If n=5 Then Exit For

Next n
```

The Do ….Until Loop repeats a statement or group of statements until a condition is met.

There are 2 ways a Do Until loop can be used in Excel VBA Macro code.

a Test the condition before executing the code in the loop

b Execute the code in the loop and then test for the condition.

Do Until….. Loop

In this example, the value of n is tested before going into the loop.

If the condition n=10 is not met right at the beginning itself, the code inside the loop is not executed at all. The control then jumps to the statements appearing after the Loop statement.

```
Do Until n=10

Debug.print n

n=n+1

Loop
```

Do ….. Loop Until

In this example, the code in the loop is executed at least once before testing the condition. If the condition is true, the looping stops, else the loop is executed again.

```
Do

Debug. print n

n=n+1

Loop Until n=10
```

The Do While … Loop repeats a statement or group of statements as long as the condition is true.

Like the Do until loop, a Do While loop can be also be used in two ways.

a Test the condition before executing the code in the loop

b Execute the code in the loop and then test for the condition.

Do While ….Loop

In this example, the condition ie. num<10 is checked before entering the loop. Only if the condition is met, the code in the loop is executed, otherwise it is skipped entirely. This example will print a table as shown in Fig 1.

Fig 1

number	spc(3)
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

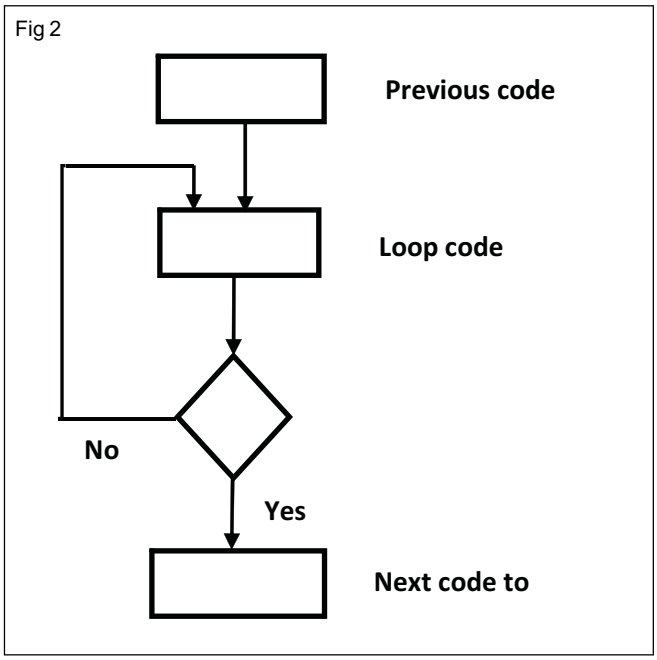
```
Dim num As Integer

Debug.Print "number"; Spc(2); "square"

Do While num < 10

num = num + 1

Debug.Print num; Spc(5); num * num

Loop
```

Do…. Loop While

In a Do…. Loop While , a set of statements in the loop are executed once, then the condition is checked. The code in the loop is executed only if the condition is met. (Refer Fig 2 for the flow chart)

In this example, the value 1 is placed in cell (1,1). The row value is incremented each time the loop code is executed. The incremented value is placed in the cell (row,1). The loop is executed as long as the row value is less than 10 after which the iterations stop. The condition checking is done after executing the loop code at least once. (Fig 2)

Fig 2




```
Dim row As Integer

row = 0

Do

row = row + 1

Cells(row, 1) = row

Loop While row < 10
```

The While ….. Wend loop

The While ….. Wend loop executes a series of statements as long as a given condition is True.

In this example the condition checking is done at the beginning of the loop. This code prints hello 5 times and then prints the value of the counter, ie. 5 at the end of the program.

```
Dim Counter

Counter = 0

While Counter < 5

Counter = Counter + 1

Debug.Print "hello"

Wend

Debug.Print Counter
```

The Exit Statement

The Exit Statement exits a procedure or block and transfers control immediately to the statement following the procedure call or the block definition. It may be in the form of Exit Do, Exit For, Exit While, Exit Select etc. depending on where it is being used. An example of an Exit statement is as follows:

```
Do While True

Count = Count + 1

Debug.Print Count

If Count = 5 Then

Debug.Print "stop at 5"

Exit Do

End If

Loop
```

In this example, the loop condition stops the loop when count=5.

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

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.

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.

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

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

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.

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"

Built in Functions in VBA

Objectives: At the end of this lesson you shall be able to
- **describe the math functions in VBA**
- **describe the logical functions in VBA**
- **describe the date/time functions in VBA**
- **describe the conversion functions in VBA.**

Introduction

VBA has a rich collection of built in functions that perform a variety of tasks and calculations for you. There are functions to convert data types, perform calculations on dates, perform simple to complex mathematics, make financial calculations, manage text strings, format values, and retrieve data from tables, among others. Using the VBA Built in Functions will help coding much easier for the user. We have already used many built in functions in our earlier lessons like the msgbox function and many string manipulation functions just to name a few.

MS Excel: VBA Functions (VBA Formulae) - Category wise

The commonly used VBA functions in Excel, sorted by Category are shown here.

String Functions: The string functions were already discussed in the related theory for Ex. 2.2.09

Math Functions

Table 1 : Lists some of the common Built in Functions in the Mathematical category.

Table 1

Function Name	Description
Abs	Returns the absolute value of a number.
Cos	Returns the cosine of the specified angle.
Cosh	Returns the hyperbolic cosine of the specified angle.
Exp	Returns e (the base of natural logarithms) raised to the specified power.
Fix	Returns the integer portion of a number.
Format	Takes a numeric expression and returns it as a formatted string.
Int	Returns the integer portion of a number.
Log	Returns the natural (base e) logarithm of a specified number or the logarithm of a specified number in a specified base.
Rnd	Generates a random number (integer value)
Round	Returns a Decimal or Double value rounded to the nearest integral value or to a specified number of fractional digits.
Sign	Returns an Integer value indicating the sign of a number.
Sin	Returns the sine of the specified angle.
Sqr	Returns the square root of a specified number.
Tan	Returns the tangent of the specified angle.
Val	Accepts a string as input and returns the numbers found in that string.

1 Dim a, b as integer

a=81

debug.print sqr(a)

This will display the square root of 81 ie. 9

Logical Functions

Table 2. lists some of the common Built in Functions in the Logical category.

Table 2

Function Name	Description
ISDATE	Returns TRUE if the expression is a valid date. Otherwise, it returns FALSE.
ISERROR	Checks for error values.
ISNULL	Returns TRUE if the expression is a null value. Otherwise, it returns FALSE.
ISNUMERIC	Returns TRUE if the expression is a valid number. Otherwise, it returns FALSE.

Examples:

1 Sub Button1_Click()

N = TextBox1.Text

If IsNumeric(N) = True Then

MsgBox "correct"

Else

MsgBox "Insert only numbers"

End If

This checks if the data entered in the textbox is a number or not.

2 Sub Button1_Click()

N = TextBox1.Text

If IsDate(N) = True Then

MsgBox "correct"

Else

MsgBox "Insert only dates"

End If

End Sub

This checks if the data entered in the textbox is a valid date or not.

Date / Time Functions

Table 3. lists some of the common Built in Functions in the Date / Time category.

Table 3

Function	Return Value
DATE	Returns the current system date.
DATEADD	Returns a date after which a certain time/date interval has been added.
DATEDIFF	Returns the difference between two date values, based on the interval specified.
DATEPART	Returns a specified part of a given date.
DATESERIAL	Returns a date given a year, month, and day value.
DATEVALUE	Returns the serial number of a date.
DAY	Returns the day of the month (a number from 1 to 31) given a date value.
FORMAT Dates	Takes a date expression and returns it as a formatted string.
HOUR	Returns the hour of a time value (from 0 to 23).
MINUTE	Returns the minute of a time value (from 0 to 59).
MONTH	Returns the month (a number from 1 to 12) given a date value.
MONTHNAME	Returns a string representing the month given a number from 1 to 12.
NOW	Returns the current system date and time.
TIMESERIAL	Returns a time given an hour, minute, and second value.
TIMEVALUE	Returns the serial number of a time.
WEEKDAY	Returns a number representing the day of the week, given a date value.
WEEKDAYNAME	Returns a string representing the day of the week given a number from 1 to 7.
YEAR	Returns the year portion of the date argument.

Examples

1 DateDiff() Function

Syntax for the DateDiff function is :

DateDiff (interval, date1, date2, [firstdayofweek], [firstweekofyear])

Parameters or Arguments

Interval is the interval of time to use to calculate the difference between date1 and date2. Below is a list of valid interval values as in Table 4

Table 4

Interval	Explanation
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week
h	Hour
n	Minute
s	Second

Date1 and Date2 are the two dates to calculate the difference between.

first day of week is optional. It is a constant that specifies the first day of the week. If this parameter is omitted, Excel assumes that Sunday is the first day of the week.

first week of year is optional. It is a constant that specifies the first week of the year. If this parameter is omitted, Excel assumes that the week containing Jan 1st is the first week of the year.

```
Sub test()

Debug.PrintDateDiff("yyyy", "1/12/1999", "31/1/2000")

Debug.PrintDateDiff("q", "1/12/1999", "31/1/2000")

Debug.PrintDateDiff("m", "1/12/1999", "31/1/2000")

End Sub
```

The result will be

1

4

12

2 Format Date

Syntax

The syntax for the Microsoft Excel FORMAT function is:

Format (expression, [format, [firstdayofweek, [firstweekofyear]]])

Parameters or Arguments

Expression is the value to format.

Format is optional. It is the format to apply to the expression. You can either define your own format or use one of the named formats that Excel has predefined such as shown in Table 5.

Table 5

Format	Explanation
General Date	Displays date based on your system settings
Long Date	Displays date based on your system's long date setting
Medium Date	Displays date based on your system's medium date setting
Short Date	Displays date based on your system's short date setting
Long Time	Displays time based on your system's long time setting
Medium Time	Displays time based on your system's medium time setting
Short Time	Displays time based on your system's short time setting

First day of week is optional. It is a value that specifies the first day of the week. If this parameter is omitted, the FORMAT function assumes that Sunday is the first day of the week. This parameter can be one of the following values as shown in Table 6.

First week of year is optional. It is a value that specifies the first week of the year. If this parameter is omitted, the FORMAT function assumes that the week that contains January 1 is the first week of the year. This parameter can be one of the following values as shown in Table 7.

```
Sub test()

Debug.Print Format(#1/1/1990#, "Short Date")

Debug.Print Format(#1/1/1990#, "Long Date")

Debug.Print Format(#1/1/1990#, "yyyy/mm/dd")

End Sub
```

Table 6

Constant	Value	Explanation
vbUseSystem	0	Uses the NLS API setting
VbSunday	1	Sunday (default, if parameter is omitted)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

The result will be

1/1/1990

Table 7

Constant	Value	Explanation
vbUseSystem	0	Uses the NLS API setting
vbFirstJan1	1	The week that contains January 1
vbFirstFourDays	2	The first week that has at least 4 days in the year
vbFirstFullWeek	3	The first full week of the year

Monday, January 01, 1990

1990/01/01

Data Type Conversion Functions

Table 8. below lists some of the common Built in Functions in the Data Type Conversion category.

Table 8

Function	Return Type	Range for expression argument
CBool	Boolean	Any valid string or numeric expression.
CByte	Byte	0 to 255.
CCur	Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807.
CDate	Date	Any valid date expression.
CDbl	Double	-1.79769313486231E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E- 324 to 1.79769313486232E308 for positive values.
CDec	Decimal	+/-79,228,162,514,264,337,593,543,950,335 for zero-scaled numbers, that is, numbers with no decimal places. For numbers with 28 decimal places, the range is +/-7.9228162514264337593543950335. The smallest possible non-zero number is 0.0000000000000000000000000001.
CInt	Integer	-32,768 to 32,767; fractions are rounded.
CLng	Long	-2,147,483,648 to 2,147,483,647; fractions are rounded.
CSng	Single	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values.
CStr	String	Returns for CStr depend on the expression argument.
CVar	Variant	Same range as Double for numerics. Same range as String for non-numerics.

Example

CDate function

```
Sub test()
Dim INum As Long
Dim a As String
a = 12345
Debug.PrintCDate(a)
b = "January 1, 1990"
Debug.PrintCDate(b)
c = "1:23:45 PM"
Debug.PrintCDate(c)
End Sub
```

This will display

10/18/1933

1/1/1990

1:23:45 PM