Aaron Wirth

Complete List Of Visual Basic Commands

Note: To go to pages use the ***Page*** function in the **View – Go To** menu, shortcut: **Shift + Ctrl + N**. Press **Ctrl + Home** to return to Table of Contents.

Table of Contents

Miscellaneous (P50)

Values (P58)

Loops and Conditional (P59)

Strings

Left and Right functions

Returns a **Variant** (**String**) containing a specified number of characters from the right side of a string.

Syntax

Left(*string, length*)
Right(*string, length*)

Example:

```
Dim AnyString, MyStr
AnyString = "Hello World"    ' Define string.
MyStr = Right(AnyString, 1)    ' Returns "d".
MyStr = Right(AnyString, 6)    ' Returns " World".
MyStr = Right(AnyString, 20)    ' Returns "Hello World".
```

Part	Description
string	Required. String expression from which the rightmost characters are returned. If ***string*** contains Null, **Null** is returned.
length	Required; **Variant** (**Long**). Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in ***string***, the entire string is returned.

Base 0 & 1

Option Base {0 | 1}

Because the default base is **0**, the **Option Base** statement is never required. If used, the statement must appear in a module before any procedures. **Option Base** can appear only once in a module and must precede array declarations that include dimensions.

The **Option Base** statement only affects the lower bound of arrays in the module where the statement is located.

Example:

```
Dim iNumber(15 To 114) As Integer
```

Trim, LTrim, and RTrim functions

Returns a **Variant** (**String**) containing a copy of a specified string without leading spaces (**LTrim**), trailing spaces (**RTrim**), or both leading and trailing spaces (**Trim**).

The required *string* argument is any valid string expression. If *string* contains Null, **Null** is returned.

Syntax

LTrim(*string*)
RTrim(*string*)
Trim(*string*)

Example:

```
Dim MyString, TrimString
MyString = "  <-Trim->  "    ' Initialize string.
TrimString = LTrim(MyString)    ' TrimString = "<-Trim->  ".
TrimString = RTrim(MyString)    ' TrimString = "  <-Trim->".
TrimString = LTrim(RTrim(MyString))    ' TrimString = "<-Trim->".
' Using the Trim function alone achieves the same result.
TrimString = Trim(MyString)    ' TrimString = "<-Trim->".
```

LCase and UCase functions

Returns a String that has been converted to lowercase.
The required *string* argument is any valid string expression. If *string* contains Null, Null is returned.

Syntax

UCase(*string*)
LCase(*string*)

Remarks

Only uppercase letters are converted to lowercase; all lowercase letters and nonletter characters remain unchanged.

Formatting

Returns a **Variant** (**String**) containing an expression formatted according to instructions contained in a format expression.

Syntax

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

The **Format** function syntax has these parts:

Part	Description
expression	Required. Any valid expression.
format	Optional. A valid named or user-defined format expression.
firstdayofweek	Optional. A constant that specifies the first day of the week.
firstweekofyear	Optional. A constant that specifies the first week of the year.

Settings

The *firstdayofweek* argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use NLS API setting.
VbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

The *firstweekofyear* argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use NLS API setting.
vbFirstJan1	1	Start with week in which January 1 occurs (default).
vbFirstFourDays	2	Start with the first week that has at least four days in the year.

vbFirstFullWeek	3	Start with the first full week of the year.

Symbol	**Range**
d	1-30
dd	1-30
ww	1-51
mmm	Displays full month names (Hijri month names have no abbreviations).
y	1-355
yyyy	100-9666

Example:
MyTime and MyDate are displayed in the development environment using current system short time setting and short date setting.

```
Dim MyTime, MyDate, MyStr
MyTime = #17:04:23#
MyDate = #January 27, 1993#

' Returns current system time in the system-defined long time format.
MyStr = Format(Time, "Long Time")

' Returns current system date in the system-defined long date format.
MyStr = Format(Date, "Long Date")

MyStr = Format(MyTime, "h:m:s")    ' Returns "17:4:23".
MyStr = Format(MyTime, "hh:mm:ss AMPM")    ' Returns "05:04:23 PM".
MyStr = Format(MyDate, "dddd, mmm d yyyy")    ' Returns "Wednesday,
   ' Jan 27 1993".
' If format is not supplied, a string is returned.
MyStr = Format(23)    ' Returns "23".

' User-defined formats.
MyStr = Format(5459.4, "##,##0.00")    ' Returns "5,459.40".
MyStr = Format(334.9, "###0.00")    ' Returns "334.90".
MyStr = Format(5, "0.00%")    ' Returns "500.00%".
MyStr = Format("HELLO", "<")    ' Returns "hello".
MyStr = Format("This is it", ">")    ' Returns "THIS IS IT".
```

FormatCurrency, FormatPercent, FormatNumber

Syntax

FormatCurrency(Expression[**,**NumDigitsAfterDecimal [**,**IncludeLeadingDigit [**,**UseParensForNegativeNumbers [**,**GroupDigits]]]]**)**
FormatPercent(Expression[**,**NumDigitsAfterDecimal [**,**IncludeLeadingDigit [**,**UseParensForNegativeNumbers [**,**GroupDigits]]]]**)** **FormatNumber(**Expression[**,**NumDigitsAfterDecimal [**,**IncludeLeadingDigit [**,**UseParensForNegativeNumbers [**,**GroupDigits]]]]**)**

Part	**Description**
Expression	Required. Expression to be formatted.

NumDigitsAfterDecimal	Optional. Numeric value indicating how many places to the right of the decimal are displayed. Default value is –1, which indicates that the computer's regional settings are used.
IncludeLeadingDigit	Optional. Tristate constant that indicates whether or not a leading zero is displayed for fractional values. See Settings section for values.
UseParensForNegativeNumbers	Optional. Tristate constant that indicates whether or not to place negative values within parentheses. See Settings section for values.
GroupDigits	Optional. Tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the computer's regional settings. See Settings section for values.

Example:
result = FormatCurrency(324.45)
result = FormatPercent(324.45, 0)
result = FormatNumber(324.45, 2)

FormatDateTime

Description
Returns an expression formatted as a date or time.
Syntax
FormatDateTime(*Date*[,*NamedFormat*])

The **FormatDateTime** function syntax has these parts:

Part	Description
Date	Required. Date expression to be formatted.
NamedFormat	Optional. Numeric value that indicates the date/time format used. If omitted, **vbGeneralDate** is used.

Settings
The *NamedFormat* argument has the following settings:

Constant	Value	Description
vbGeneralDate	0	Display a date and/or time. If there is a date part, display it as a short date. If there is a time part, display it as a long time. If present, both parts are displayed.
vbLongDate	1	Display a date using the long date format specified in your computer's regional settings.
vbShortDate	2	Display a date using the short date format specified in your computer's regional settings.
vbLongTime	3	Display a time using the time format specified in your computer's regional settings.
vbShortTime	4	Display a time using the 24-hour format (hh:mm).

Mid function

Returns a **Variant** (**String**) containing a specified number of characters from a string.
To determine the number of characters in ***string***, use the **Len** function.
Syntax
Mid(*string*, *start*[, *length*])

The **Mid** function syntax has these named arguments:

Part	Description
string	Required. String expression from which characters are returned. If ***string*** contains Null, **Null** is returned.
start	Required; Long. Character position in ***string*** at which the part to be taken begins. If ***start*** is greater than the number of characters in ***string***, **Mid** returns a zero-length string ("").
length	Optional; **Variant** (**Long**). Number of characters to return. If omitted or if there are fewer than ***length*** characters in the text (including the character at ***start***), all characters from the ***start*** position to the end of the string are returned.

Example:

```
Dim MyString, FirstWord, LastWord, MidWords
MyString = "Mid Function Demo"    ' Create text string.
FirstWord = Mid(MyString, 1, 3)    ' Returns "Mid".
LastWord = Mid(MyString, 14, 4)    ' Returns "Demo".
MidWords = Mid(MyString, 5)    ' Returns "Function Demo".
```

Chr Function

Returns a String containing the character associated with the specified character code.
The required *charcode* argument is a Long that identifies a character.
Syntax
Chr(*charcode*)

Example:

```
Dim MyChar
MyChar = Chr(65)    ' Returns A.
MyChar = Chr(97)    ' Returns a.
MyChar = Chr(62)    ' Returns >.
MyChar = Chr(37)    ' Returns %.
```

Len Function

Returns a Long containing the number of characters in a string or the number of bytes required to store a variable.
Syntax
Len(*string* | *varname*)

The **Len** function syntax has these parts:

Part	Description
string	Any valid string expression. If *string* contains Null, Null is returned.
Varname	Any valid variable name. If *varname* contains **Null**, **Null** is returned. If *varname* is a Variant, **Len** treats it the same as a **String** and always returns the number of characters it contains.

Example:

```
Label1 = Len(Text1)  ' Text1 = “Blah”
                     ' Label1 = “4”
```

InStr function

Returns a **Variant** (**Long**) specifying the position of the first occurrence of one string within another.

Syntax

InStr([*start*,]*string1*, *string2*[, *compare*])

The **InStr** function syntax has these arguments:

Part	**Description**
start	Optional. Numeric expression that sets the starting position for each search. If omitted, search begins at the first character position. If ***start*** contains Null, an error occurs. The ***start*** argument is required if ***compare*** is specified.
string1	Required. String expression being searched.
string2	Required. String expression sought.
compare	Optional. Specifies the type of string comparison. If ***compare*** is Null, an error occurs. If ***compare*** is omitted, the **Option Compare** setting determines the type of comparison. Specify a valid LCID (LocaleID) to use locale-specific rules in the comparison.

Settings

The *compare* argument settings are:

Constant	Value	Description
vbUseCompareOption	-1	Performs a comparison using the setting of the Option Compare statement.
vbBinaryCompare	0	Performs a binary comparison.
vbTextCompare	1	Performs a textual comparison.
vbDatabaseCompare	2	Microsoft Access only. Performs a comparison based on information in your database.

Return Values

If	InStr returns
string1 is zero-length	0
string1 is Null	Null
string2 is zero-length	*start*
string2 is Null	Null
string2 is not found	0
string2 is found within *string1*	Position at which match is found
start > *string2*	0

Example:

```
Dim SearchString, SearchChar, MyPos
SearchString ="XXpXXpXXPXXP"    ' String to search in.
SearchChar = "P"    ' Search for "P".
' A textual comparison starting at position 4. Returns 6.
MyPos = Instr(4, SearchString, SearchChar, 1)
' A binary comparison starting at position 1. Returns 9.
MyPos = Instr(1, SearchString, SearchChar, 0)
MyPos = Instr(SearchString, SearchChar)    ' Returns 9.
```

MyPos = **Instr(**1**,** SearchString**,** "W"**)** ' Returns 0.

String function

Returns a **Variant** (**String**) containing a repeating character string of the length specified.
Syntax
String(***number***, ***character***)

The **String** function syntax has these named arguments:

Part	**Description**
number	Required; Long. Length of the returned string. If ***number*** contains Null, **Null** is returned.
character	Required; Variant. Character code specifying the character or string expression whose first character is used to build the return string. If ***character*** contains **Null**, **Null** is returned.

Remarks
If you specify a number for ***character*** greater than 255, **String** converts the number to a valid character code using the formula:
character **Mod** 256

InstrRev

Returns the position of an occurrence of one string within another, from the end of string.
Syntax
InstrRev(*stringcheck*, *stringmatch*[, *start*[, *compare*]])

Settings
The *compare* argument can have the following values:

Constant	Value	Description
vbUseCompareOption	–1	Performs a comparison using the setting of the Option Compare statement.
vbBinaryCompare	0	Performs a binary comparison.
vbTextCompare	1	Performs a textual comparison.
vbDatabaseCompare	2	Microsoft Access only. Performs a comparison based on information in your database.

Asc Function

Returns an Integer representing the character code corresponding to the first letter in a string.
Syntax
Asc(*string***)**

The required *string* argument is any valid string expression. If the *string* contains no characters, a run-time error occurs.

Example:
Dim MyNumber
MyNumber = **Asc(**"A"**)** ' Returns 65.
MyNumber = **Asc(**"a"**)** ' Returns 97.
MyNumber = **Asc(**"Apple"**)** ' Returns 65.

Space Function

This function by itself produces a certain number of spaces. It's best use is to clear fixed-length strings.
sRecord$ = Space(128)

Replace Function

Returns a string in which a specified substring has been replaced with another substring a specified number of times.

Syntax

Replace(*expression*, *find*, *replace*[, *start*[, *count*[, *compare*]]])

The Replace function syntax has these named arguments:

Part	Description
expression	Required. String expression containing substring to replace.
find	Required. Substring being searched for.
replace	Required. Replacement substring.
start	Optional. Position within *expression* where substring search is to begin. If omitted, 1 is assumed.
count	Optional. Number of substring substitutions to perform. If omitted, the default value is –1, which means make all possible substitutions.
compare	Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. See Settings section for values.

StrComp function

Returns a **Variant** (**Integer**) indicating the result of a string comparison.

Syntax

StrComp(***string1***, ***string2***[, ***compare***])

The **StrComp** function syntax has these named arguments:

Part	**Description**
string1	Required. Any valid string expression.
string2	Required. Any valid string expression.
compare	Optional. Specifies the type of string comparison. If the ***compare*** argument is Null, an error occurs. If ***compare*** is omitted, the **Option Compare** setting determines the type of comparison.

Example:

```
Dim MyStr1, MyStr2, MyComp
MyStr1 = "ABCD": MyStr2 = "abcd"    ' Define variables.
MyComp = StrComp(MyStr1, MyStr2, 1)    ' Returns 0.
MyComp = StrComp(MyStr1, MyStr2, 0)    ' Returns -1.
MyComp = StrComp(MyStr2, MyStr1)    ' Returns 1.
```

StrConv function

Returns a **Variant** (**String**) converted as specified.

Syntax

StrConv(***string, conversion, LCID***)

The **StrConv** function syntax has these named arguments:

Part	**Description**
string	Required. String expression to be converted.

conversion	Required. Integer. The sum of values specifying the type of conversion to perform.
LCID	Optional. The LocaleID, if different than the system LocaleID. (The system LocaleID is the default.)

Math

Val function

Returns the numbers contained in a string as a numeric value of appropriate type.

Syntax

Val(*string***)**

The required *string* argument is any valid string expression.

The **Val** function stops reading the string at the first character it can't recognize as part of a number. Symbols and characters that are often considered parts of numeric values, such as dollar signs and commas, are not recognized. However, the function recognizes the radix prefixes &O (for octal) and &H (for hexadecimal). Blanks, tabs, and linefeed characters are stripped from the argument.

Round

Description

Returns a number rounded to a specified number of decimal places.

Syntax

Round(*expression* [,*numdecimalplaces*])

The Round function syntax has these parts:

Part	Description
expression	Required. Numeric expression being rounded.
numdecimalplaces	Optional. Number indicating how many places to the right of the decimal are included in the rounding. If omitted, integers are returned by the Round function.

Example:

```
Text1.Text = Number
Round(Number,5)
‘Rounds the number in text1 to 5 decimal places
```

Int and Fix functions

Returns the integer portion of a number.

Syntax

Int(*number***)**

Fix(*number***)**

The required *number* argument is a Double or any valid numeric expression. If *number* contains Null, **Null** is returned.

Both **Int** and **Fix** remove the fractional part of *number* and return the resulting integer value.

The difference between **Int** and **Fix** is that if *number* is negative, **Int** returns the first negative integer less than or equal to *number*, whereas **Fix** returns the first negative integer greater than or equal to *number*. For example, **Int** converts -8.4 to -9, and **Fix** converts -8.4 to -8.

Rnd and Randomize functions

A function which generates a random number.

Randomize uses *number* to initialize the **Rnd** function's random-number generator, giving it a new seed value. If you omit *number*, the value returned by the system timer is used as the new seed value.

Syntax

Randomize [*number*]

Rnd[**(***number***)**]

Example:

```
Randomize
Label1 = Int((6 * Rnd) + 1)    'Generate random value between 1 and 6.
```

Sgn function

Returns a **Variant** (**Integer**) indicating the sign of a number.
Syntax
Sgn(*number***)**

The required *number* argument can be any valid numeric expression.
Return Values

If *number* is	**Sgn returns**
Greater than zero	1
Equal to zero	0
Less than zero	-1

Example:

```
Dim MyVar1, MyVar2, MyVar3, MySign
MyVar1 = 12: MyVar2 = -2.4: MyVar3 = 0
MySign = Sgn(MyVar1)    ' Returns 1.
MySign = Sgn(MyVar2)    ' Returns -1.
MySign = Sgn(MyVar3)    ' Returns 0.
```

Sin, Cos, Tan, Log, Atn & Exp Functions

If you're into geometry, you're all set there too. From the list of VB functions below, you can make any geometric calculation that exists. (Assuming you're Albert Einstein).
Syntax
Sin(*number***)**
Cos(*number***)**
Tan(*number***)**
Log(*number***)**
Atn(*number***)**
Exp(*number***)**

Abs function

Returns a value of the same type that is passed to it specifying the absolute value of a number.
Syntax
Abs(*number***)**

The required *number* argument can be any valid numeric expression. If *number* contains Null, **Null** is returned; if it is an uninitialized variable, zero is returned.

Example:

```
Dim MyNumber
MyNumber = Abs(50.3)    ' Returns 50.3.
MyNumber = Abs(-50.3)    ' Returns 50.3.
```

Other Math Functions

The following is a list of nonintrinsic math functions that can be derived from the intrinsic math functions:

Function	Derived equivalents
Secant	Sec(X) = 1 / Cos(X)
Cosecant	Cosec(X) = 1 / Sin(X)
Cotangent	Cotan(X) = 1 / Tan(X)
Inverse Sine	Arcsin(X) = Atn(X / Sqr(-X * X + 1))
Inverse Cosine	Arccos(X) = Atn(-X / Sqr(-X * X + 1)) + 2 * Atn(1)
Inverse Secant	Arcsec(X) = Atn(X / Sqr(X * X – 1)) + Sgn((X) – 1) * (2 * Atn(1))
Inverse Cosecant	Arccosec(X) = Atn(X / Sqr(X * X - 1)) + (Sgn(X) – 1) * (2 * Atn(1))
Inverse Cotangent	Arccotan(X) = Atn(X) + 2 * Atn(1)
Hyperbolic Sine	HSin(X) = (Exp(X) – Exp(-X)) / 2
Hyperbolic Cosine	HCos(X) = (Exp(X) + Exp(-X)) / 2
Hyperbolic Tangent	HTan(X) = (Exp(X) – Exp(-X)) / (Exp(X) + Exp(-X))
Hyperbolic Secant	HSec(X) = 2 / (Exp(X) + Exp(-X))
Hyperbolic Cosecant	HCosec(X) = 2 / (Exp(X) – Exp(-X))
Hyperbolic Cotangent	HCotan(X) = (Exp(X) + Exp(-X)) / (Exp(X) – Exp(-X))
Inverse Hyperbolic Sine	HArcsin(X) = Log(X + Sqr(X * X + 1))
Inverse Hyperbolic Cosine	HArccos(X) = Log(X + Sqr(X * X – 1))
Inverse Hyperbolic Tangent	HArctan(X) = Log((1 + X) / (1 – X)) / 2
Inverse Hyperbolic Secant	HArcsec(X) = Log((Sqr(-X * X + 1) + 1) / X)
Inverse Hyperbolic Cosecant	HArccosec(X) = Log((Sgn(X) * Sqr(X * X + 1) + 1) / X)
Inverse Hyperbolic Cotangent	HArccotan(X) = Log((X + 1) / (X – 1)) / 2
Logarithm to base N	LogN(X) = Log(X) / Log(N)

Logic

Mod Operator

Syntax

result = *number1* Mod *number2*

The Mod operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
number1	Required; any numeric expression.
number2	Required; any numeric expression.

Remarks

The modulus, or remainder, operator divides *number1* by *number2* (rounding floating-point numbers to integers) and returns only the remainder as *result*. For example, in the following expression, A (*result*) equals 5.
A = 19 Mod 6.7

Example:

```
Dim MyResult
MyResult = 10 Mod 5    ' Returns 0.
MyResult = 10 Mod 3    ' Returns 1.
MyResult = 12 Mod 4.3    ' Returns 0.
MyResult = 12.6 Mod 5    ' Returns 3.
```

And Operator

Used to perform a logical conjunction on two expressions..

Syntax

result = *expression1* **And** *expression2*

The **And** operator syntax has these parts:

Part	**Description**
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If both expressions evaluate to **True**, *result* is **True**. If either expression evaluates to **False**, *result* is **False.** The following table illustrates how *result* is determined:

If *expression1* is	**And *expression2* is**	**The *result* is**
True	**True**	**True**
True	**False**	**False**
True	Null	**Null**
False	**True**	**False**
False	**False**	**False**
False	**Null**	**False**
Null	**True**	**Null**

Null	**False**	**False**
Null	**Null**	**Null**

Example:
```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null    ' Initialize variables.
MyCheck = A > B And B > C    ' Returns True.
MyCheck = B > A And B > C    ' Returns False.
MyCheck = A > B And B > D    ' Returns Null.
```

Or Operator

Used to perform a logical disjunction on two expressions.

Syntax

result = *expression1* **Or** *expression2*

The **Or** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If either or both expressions evaluate to **True**, *result* is **True**. The following table illustrates how *result* is determined:

If *expression1* is	And *expression2* is	Then *result* is
True	**True**	**True**
True	**False**	**True**
True	Null	**True**
False	**True**	**True**
False	**False**	**False**
False	**Null**	**Null**
Null	**True**	**True**
Null	**False**	**Null**
Null	**Null**	**Null**

Example:
```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null    ' Initialize variables.
MyCheck = A > B Or B > C    ' Returns True.
MyCheck = B > A Or B > C    ' Returns True.
MyCheck = A > B Or B > D    ' Returns True.
MyCheck = B > D Or B > A    ' Returns Null.
```

Xor Operator

Used to perform a logical exclusion on two expressions.

Syntax

[result =] expression1 Xor expression2

The Xor operator syntax has these parts:

Part	Description
result	Optional; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If one, and only one, of the expressions evaluates to True, result is True. However, if either expression is Null, result is also Null. When neither expression is Null, result is determined according to the following table:

If expression1 is	And expression2 is	Then result is
True	True	False
True	False	True
False	True	True
False	False	False

Example:

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null    ' Initialize variables.
MyCheck = A > B Xor B > C    ' Returns False.
MyCheck = B > A Xor B > C    ' Returns True.
MyCheck = B > A Xor C > B    ' Returns False.
MyCheck = B > D Xor A > B    ' Returns Null.
```

If Not

If Not is the exact opposite of If, the code segment will run if a condition is False.

Example:

```
Dim Done As Boolean
Done = True
If Not Done Then LetsFinish
```

The LetsFinish procedure will not run. Notice we just used the boolean variable by itself. **If Not Done** is equivalent to **If Done = False** and **If Done** is the same as **If Done = True**.

Like operator

Used to compare two strings.

Syntax

result = *string* **Like** *pattern*

The **Like** operator syntax has these parts:

Part	**Description**
result	Required; any numeric variable.
string	Required; any string expression.
pattern	Required; any string expression conforming to the pattern-matching conventions described in Remarks.

Remarks
If *string* matches *pattern*, *result* is **True**; if there is no match, *result* is **False**. If either *string* or *pattern* is Null, *result* is **Null**.
The behavior of the **Like** operator depends on the **Option Compare** statement. The default string-comparison method for each module is **Option Compare Binary**.
Built-in pattern matching provides a versatile tool for string comparisons. The pattern-matching features allow you to use wildcard characters, character lists, or character ranges, in any combination, to match strings. The following table shows the characters allowed in *pattern* and what they match:

Characters in *pattern*	**Matches in *string***
?	Any single character.
*****	Zero or more characters.
#	Any single digit (0–9).
[*charlist*]	Any single character in *charlist*.
[**!***charlist*]	Any single character not in *charlist*.

Example:
```
Dim Name As String
Name = InputBox("Enter your name")
Find out if user's name begins with a J
If sName$ Like "J*" Then
    (code segment)
End If
```

Is Operator
Used to compare two object reference variables.
Syntax
result **=** *object1* **Is** *object2*
The **Is** operator syntax has these parts:

Part	**Description**
result	Required; any numeric variable.
object1	Required; any object name.
object2	Required; any object name.

Remarks
If *object1* and *object2* both refer to the same object, *result* is **True**; if they do not, *result* is **False**. Two variables can be made to refer to the same object in several ways.

Example:
```
MyCheck = YourObject Is ThisObject    ' Returns True.
MyCheck = ThatObject Is ThisObject    ' Returns False.
```

Arrays

Erase statement

Reinitializes the elements of fixed-size arrays and releases dynamic-array storage space.

Syntax

Erase *arraylist*

The required *arraylist* argument is one or more comma-delimited array variables to be erased.

Remarks

Erase behaves differently depending on whether an array is fixed-size (ordinary) or dynamic. **Erase** recovers no memory for fixed-size arrays. **Erase** sets the elements of a fixed array as follows:

Type of Array	Effect of Erase on Fixed-Array Elements
Fixed numeric array	Sets each element to zero.
Fixed string array (variable length)	Sets each element to a zero-length string ("").
Fixed string array (fixed length)	Sets each element to zero.
Fixed Variant array	Sets each element to Empty.
Array of user-defined types	Sets each element as if it were a separate variable.
Array of objects	Sets each element to the special value **Nothing**.

Example:

Erase sMessage

In a regular array, the Erase statement will simply initialize all the elements. (False for Boolean, 0 for numbers, and "" for strings). In a dynamic array, Erase will also release all the memory allocated to the array.

Dim

Dim statement placed right in the procedure where it's going to be used. The value of a procedure level variable cannot be accessed outside it's procedure. When the procedure finishes (End Sub or End Function), the variable is destroyed and memory allocated to the variable is released.

Example:

Dim Word As String

ReDim

Used at procedure level to reallocate storage space for dynamic array variables.

Syntax

ReDim [**Preserve**] *varname***(***subscripts***)** [**As** *type*] [**,** *varname***(***subscripts***)** [**As** *type*]] **. . .**

The **ReDim** statement syntax has these parts:

Part	Description
Preserve	Optional. Keyword used to preserve the data in an existing array when you change the size of the last dimension.
varname	Required. Name of the variable; follows standard variable naming conventions.
subscripts	Required. Dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax: [*lower* **To**] *upper* [**,**[*lower* **To**] *upper*] **. . .** When not explicitly stated in *lower*, the lower bound of an array is controlled by the **Option Base** statement. The lower bound is zero if no **Option Base** statement is present.

type	Optional. Data type of the variable; may be Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal (not currently supported), Date, String (for variable-length strings), **String** * *length* (for fixed-length strings), Object, Variant, a user-defined type, or an object type. Use a separate **As** *type* clause for each variable being defined. For a **Variant** containing an array, *type* describes the type of each element of the array, but doesn't change the **Variant** to some other type.

Remarks
The **ReDim** statement is used to size or resize a dynamic array that has already been formally declared using a **Private**, **Public**, or **Dim** statement with empty parentheses (without dimension subscripts).

Example:
Dim X(10, 10, 10)
[Code]
ReDim Preserve X(10, 10, 15)

Array Function

Returns a Variant containing an array.
Syntax
Array(*arglist***)**

The required *arglist* argument is a comma-delimited list of values that are assigned to the elements of the array contained within the **Variant**. If no arguments are specified, an array of zero length is created.
Remarks
The notation used to refer to an element of an array consists of the variable name followed by parentheses containing an index number indicating the desired element. In the following example, the first statement creates a variable named A as a **Variant**. The second statement assigns an array to variable A. The last statement assigns the value contained in the second array element to another variable.

Example:
Dim MyWeek, MyDay
MyWeek = **Array(**"Mon"**,** "Tue"**,** "Wed"**,** "Thu"**,** "Fri"**,** "Sat"**,** "Sun"**)**
' Return values assume lower bound set to 1 (using Option Base
' statement).
MyDay = MyWeek(2) ' MyDay contains "Tue".
MyDay = MyWeek(4) ' MyDay contains "Thu".

Files/Folders

Dir

Returns a **String** representing the name of a file, directory, or folder that matches a specified pattern or file attribute, or the volume label of a drive.

Syntax

Dir[**(***pathname*[**,** *attributes*]**)**]

The **Dir** function syntax has these parts:

Part	Description
pathname	Optional. String expression that specifies a file name — may include directory or folder, and drive. A zero-length string ("") is returned if *pathname* is not found.
attributes	Optional. Constant or numeric expression, whose sum specifies file attributes. If omitted, returns files that match *pathname* but have no attributes.

Settings

The *attributes* argument settings are:

Constant	Value	Description
vbNormal	0	(Default) Specifies files with no attributes.
vbReadOnly	1	Specifies read-only files in addition to files with no attributes.
vbHidden	2	Specifies hidden files in addition to files with no attributes.
VbSystem	4	Specifies system files in addition to files with no attributes. Not available on the Macintosh.
vbVolume	8	Specifies volume label; if any other attributed is specified, **vbVolume** is ignored. Not available on the Macintosh.
vbDirectory	16	Specifies directories or folders in addition to files with no attributes.
vbAlias	64	Specified file name is an alias. Available only on the Macintosh.

Note These constants are specified by Visual Basic for Applications and can be used anywhere in your code in place of the actual values.

Example:

```
If Dir("c:\windows\win.ini") = "win.ini" Then
    MsgBox "File exists"
Else
    MsgBox "File does not exist"
End If
```

ChDir
Changes the current directory or folder.
Syntax
ChDir *path*

The required *path* argument is a string expression that identifies which directory or folder becomes the new default directory or folder. The *path* may include the drive. If no drive is specified, **ChDir** changes the default directory or folder on the current drive.
Remarks
The **ChDir** statement changes the default directory but not the default drive. For example, if the default drive is C, the following statement changes the default directory on drive D, but C remains the default drive.
Example:
Dim Path as string
Dir(“C:\NewFolder”) = Path
Path = **ChDir**(“C:\MyFolder”)

ChDrive
Changes the current drive.
Syntax
ChDrive *drive*

The required *drive* argument is a string expression that specifies an existing drive. If you supply a zero-length string (""), the current drive doesn't change. If the *drive* argument is a multiple-character string, **ChDrive** uses only the first letter.

Example:
ChDrive "D" ' Make "D" the current drive.

CurDir
Returns a **Variant** (**String**) representing the current path.
Syntax
CurDir[**(***drive***)**]

The optional *drive* argument is a string expression that specifies an existing drive. If no drive is specified or if *drive* is a zero-length string (""), **CurDir** returns the path for the current drive. On the Macintosh, **CurDir** ignores any *drive* specified and simply returns the path for the current drive.

Example:
' Assume current path on C drive is "C:\WINDOWS\SYSTEM" (on Microsoft Windows).
' Assume current path on D drive is "D:\EXCEL".
' Assume C is the current drive.
Dim MyPath
MyPath = **CurDir** ' Returns "C:\WINDOWS\SYSTEM".
MyPath = **CurDir(**"C"**)** ' Returns "C:\WINDOWS\SYSTEM".
MyPath = **CurDir(**"D"**)** ' Returns "D:\EXCEL".

MkDir
Creates a new directory or folder.
Syntax
MkDir *path*

The required *path* argument is a string expression that identifies the directory or folder to be created. The *path* may include the drive. If no drive is specified, **MkDir** creates the new directory or folder on the current drive.

Example:
‘Creates a folder called ‘New Folder’
MkDir “C:\New Folder”

RmDir Function

Removes an existing directory or folder.

Syntax

RmDir *path*

The required *path* argument is a string expression that identifies the directory or folder to be removed. The *path* may include the drive. If no drive is specified, **RmDir** removes the directory or folder on the current drive.

Remarks

An error occurs if you try to use **RmDir** on a directory or folder containing files. Use the **Kill** statement to delete all files before attempting to remove a directory or folder.

Example:

```
RmDir "c:\windows\pictures"
```

Kill Function

Deletes files from a disk.

Syntax

Kill *pathname*

The required *pathname* argument is a string expression that specifies one or more file names to be deleted. The *pathname* may include the directory or folder, and the drive.

Example:

```
Kill "c:\Blah.txt”
```

FileDateTime

Returns a **Variant** (**Date**) that indicates the date and time when a file was created or last modified.

Syntax

FileDateTime(*pathname***)**

The required *pathname* argument is a string expression that specifies a file name. The *pathname* may include the directory or folder, and the drive.

Example:

```
Dim MyStamp
' Assume TESTFILE was last modified on September 2, 2005 at 4:00:00 PM.
MyStamp = FileDateTime "c:\Blah.txt” ' Returns "2/9/05 4:00:00 PM".
```

FileLen

Returns a Long specifying the length of a file in bytes.

Syntax

FileLen(*pathname***)**

The required *pathname* argument is a string expression that specifies a file. The *pathname* may include the directory or folder, and the drive.

Remarks

If the specified file is open when the **FileLen** function is called, the value returned represents the size of the file immediately before it was opened.

Note To obtain the length of an open file, use the **LOF** function.

Example:

```
Dim Size
Size = FileLen("TESTFILE")    ' Returns file length (bytes).
```

FileCopy
Copies a file.
Syntax
FileCopy ***source, destination***

The **FileCopy** statement syntax has these named arguments:

Part	Description
source	Required. String expression that specifies the name of the file to be copied. The ***source*** may include directory or folder, and drive.
destination	Required. String expression that specifies the target file name. The ***destination*** may include directory or folder, and drive.

Remarks
If you try to use the **FileCopy** statement on a currently open file, an error occurs.

Cut, Copy & Pasting Text
The Clipboard object represents the Windows clipboard which is available to all running applications, therefore you can allow your users to place text or pictures on the Clipboard and paste them anywhere they like. Setting up menu items for cut, copy, and pasting from and into one TextBox is fairly simple.

Example:
```
Private Sub mnuCopy_Click( )
   Clipboard.Clear
   Clipboard.SetText txtMain.SelText
End Sub

Private Sub mnuCut_Click( )
   Clipboard.Clear
   Clipboard.SetText txtMain.SelText
   txtMain.SelText = ""
End Sub

Private Sub mnuPaste_Click( )
   txtMain.SelText = Clipboard.GetText
End Sub
```

GetAttr
Returns an **Integer** representing the attributes of a file, directory, or folder.
Syntax
GetAttr(*pathname***)**

The required *pathname* argument is a string expression that specifies a file name. The *pathname* may include the directory or folder, and the drive.
Return Values
The value returned by **GetAttr** is the sum of the following attribute values:

Constant	Value	Description
vbNormal	0	Normal.
vbReadOnly	1	Read-only.
vbHidden	2	Hidden.
vbSystem	4	System file. Not available on the Macintosh.
vbDirectory	16	Directory or folder.

vbArchive	32	File has changed since last backup. Not available on the Macintosh.
vbAlias	64	Specified file name is an alias. Available only on the Macintosh.

Note These constants are specified by Visual Basic for Applications. The names can be used anywhere in your code in place of the actual values.

SetAttr

Sets attribute information for a file.

Syntax

SetAttr ***pathname, attributes***

The **SetAttr** statement syntax has these named arguments:

Part	Description
pathname	Required. String expression that specifies a file name — may include directory or folder, and drive.
attributes	Required. Constant or numeric expression, whose sum specifies file attributes.

Settings

The ***attributes*** argument settings are:

Constant	Value	Description
vbNormal	0	Normal (default).
vbReadOnly	1	Read-only.
vbHidden	2	Hidden.
vbSystem	4	System file. Not available on the Macintosh.
vbArchive	32	File has changed since last backup.
vbAlias	64	Specified file name is an alias. Available only on the Macintosh.

FreeFile function

Returns an Integer representing the next file number available for use by the **Open** statement.

Syntax

FreeFile[**(***rangenumber***)**]

The optional *rangenumber* argument is a Variant that specifies the range from which the next free file number is to be returned. Specify a 0 (default) to return a file number in the range 1 – 255, inclusive. Specify a 1 to return a file number in the range 256 – 511.

Remarks

Use **FreeFile** to supply a file number that is not already in use.

Example:

```
Dim MyIndex, FileNumber
For MyIndex = 1 To 5    ' Loop 5 times.
   FileNumber = FreeFile    ' Get unused file
      ' number.
   Open "TEST" & MyIndex For Output As #FileNumber    ' Create file name.
   Write #FileNumber, "This is a sample."    ' Output text.
   Close #FileNumber    ' Close file.
Next MyIndex
```

Open Function

Enables input/output (I/O) to a file.

Syntax

Open *pathname* **For** *mode* [**Access** *access*] [*lock*] **As** [#]*filenumber* [**Len**=*reclength*]

The **Open** statement syntax has these parts:

Part	Description
pathname	Required. String expression that specifies a file name — may include directory or folder, and drive.
mode	Required. Keyword specifying the file mode: **Append**, **Binary**, **Input**, **Output**, or **Random**. If unspecified, the file is opened for **Random** access.
access	Optional. Keyword specifying the operations permitted on the open file: **Read**, **Write**, or **Read Write**.
lock	Optional. Keyword specifying the operations restricted on the open file by other processes: **Shared**, **Lock Read**, **Lock Write**, and **Lock Read Write**.
filenumber	Required. A valid file number in the range 1 to 511, inclusive. Use the **FreeFile** function to obtain the next available file number.
reclength	Optional. Number less than or equal to 32,767 (bytes). For files opened for random access, this value is the record length. For sequential files, this value is the number of characters buffered.

Remarks

Output: use the print statement to print something in a file.
Input: use this to open and read a file.
Append: use this to write something into the file but keep what is already in the file.
Binary: use this for binary access files
Random: use this for random access files
You must open a file before any I/O operation can be performed on it. **Open** allocates a buffer for I/O to the file and determines the mode of access to use with the buffer.
If the file specified by *pathname* doesn't exist, it is created when a file is opened for **Append**, **Binary**, **Output**, or **Random** modes.
If the file is already opened by another process and the specified type of access is not allowed, the **Open** operation fails and an error occurs.
The **Len** clause is ignored if *mode* is **Binary**.
Important In **Binary**, **Input**, and **Random** modes, you can open a file using a different file number without first closing the file. In **Append** and **Output** modes, you must close a file before opening it with a different file number.

Example:

```
Open “a:\gordon.txt” For Output As #1
    Print #1, Text1.Text ‘prints Text1 into the file
Close #1
```

Close Statement

Concludes input/output (I/O) to a file opened using the **Open** statement.

Syntax

Close [*filenumberlist*]

The optional *filenumberlist* argument can be one or more file numbers using the following syntax, where *filenumber* is any valid file number:

[[**#**]*filenumber*] [**,** [**#**]*filenumber*] **. . .**

Remarks

If you omit *filenumberlist*, all active files opened by the **Open** statement are closed.

When you close files that were opened for **Output** or **Append**, the final buffer of output is written to the operating system buffer for that file. All buffer space associated with the closed file is released.

When the **Close** statement is executed, the association of a file with its file number ends.

Line Input

Reads a single line from an open sequential file and assigns it to a String variable.

Syntax

Line Input #*filenumber*, *varname*

The **Line Input #** statement syntax has these parts:

Part	**Description**
filenumber	Required. Any valid file number.
varname	Required. Valid Variant or **String** variable name.

Remarks

Data read with **Line Input #** is usually written from a file with **Print #**.

Example:

```
Open “a:\gordon.txt” For Input As #1
    Line Input #1, TextLine ' Read line into variable.
Close #1
```

EOF Function

Returns an Integer containing the Boolean value **True** when the end of a file opened for **Random** or sequential **Input** has been reached.

Syntax

EOF(*filenumber***)**

The required *filenumber* argument is an **Integer** containing any valid file number.

Remarks

Use **EOF** to avoid the error generated by attempting to get input past the end of a file.

The **EOF** function returns **False** until the end of the file has been reached. With files opened for **Random** or **Binary** access, **EOF** returns **False** until the last executed **Get** statement is unable to read an entire record.

With files opened for **Binary** access, an attempt to read through the file using the **Input** function until **EOF** returns **True** generates an error. Use the **LOF** and **Loc** functions instead of **EOF** when reading binary files with **Input**, or use **Get** when using the **EOF** function. With files opened for **Output**, **EOF** always returns **True**.

Example:

```
dim templine as string
Open “a:\ratbag.txt” For Input As 1
  Do Until EOF(1)
    Line Input #1, templine
    Text1.Text = text1.text + templine
  Loop
Close #1
```

Lof Function

Returns a Long representing the size, in bytes, of a file opened using the **Open** statement.

Syntax

LOF(*filenumber***)**

The required *filenumber* argument is an Integer containing a valid file number.

Note Use the **FileLen** function to obtain the length of a file that is not open.

Example:

```
Dim FileLength
Open "TESTFILE" For Input As #1    ' Open file.
FileLength = LOF(1)    ' Get length of file.
Close #1    ' Close file.
```

Print Function

Writes display-formatted data to a sequential file.

Syntax

Print #*filenumber*, [*outputlist*]

The **Print #** statement syntax has these parts:

Part	Description
filenumber	Required. Any valid file number.
outputlist	Optional. Expression or list of expressions to print.

Remarks

Data written with Print # is usually read from a file with Line Input # or Input.

Example:

```
Open App.Path & "\myFile.txt" For Output As #1
    Print #1, "Blah, Blah" + Text1.Text
Close #1
```

Error Handling

On Error Statement

Enables an error-handling routine and specifies the location of the routine within a procedure; can also be used to disable an error-handling routine.

Syntax

On Error GoTo *line*
On Error Resume Next
On Error GoTo 0

The **On Error** statement syntax can have any of the following forms:

Statement	Description
On Error GoTo *line*	Enables the error-handling routine that starts at *line* specified in the required *line* argument. The *line* argument is any line label or line number. If a run-time error occurs, control branches to *line*, making the error handler active. The specified *line* must be in the same procedure as the **On Error** statement; otherwise, a compile-time error occurs.
On Error Resume Next	Specifies that when a run-time error occurs, control goes to the statement immediately following the statement where the error occurred where execution continues. Use this form rather than **On Error GoTo** when accessing objects.
On Error GoTo 0	Disables any enabled error handler in the current procedure.

Remarks

If you don't use an **On Error** statement, any run-time error that occurs is fatal; that is, an error message is displayed and execution stops.

Resume, Resume Next, Resume Line ()

Resumes execution after an error-handling routine is finished.

Syntax

Resume [**0**]
Resume Next
Resume *line*

The **Resume** statement syntax can have any of the following forms:

Statement	Description
Resume	If the error occurred in the same procedure as the error handler, execution resumes with the statement that caused the error. If the error occurred in a called procedure, execution resumes at the statement that last called out of the procedure containing the error-handling routine.
Resume Next	If the error occurred in the same procedure as the error handler, execution resumes with the statement immediately following the statement that caused the error. If the error occurred in a called procedure, execution resumes with the statement immediately following the statement that last called out of the procedure containing the error-handling routine (or **On Error Resume Next** statement).
Resume *line*	Execution resumes at *line* specified in the required *line* argument. The *line* argument is a line label or line number and must be in the same procedure as the error handler.

Remarks

If you use a **Resume** statement anywhere except in an error-handling routine, an error occurs.

Example:
```
Private Sub
        …
        On Error GoTo Error
End Sub
        Error:
        …
        Resume
```

Error Function

Simulates the occurrence of an error.

Syntax

Error *errornumber*

The required *errornumber* can be any valid error number.

Remarks

The **Error** statement is supported for backward compatibility. In new code, especially when creating objects, use the **Err** object's **Raise** method to generate run-time errors.

If *errornumber* is defined, the **Error** statement calls the error handler after the properties of **Err** object are assigned the following default values:

Property	Value
Number	Value specified as argument to **Error** statement. Can be any valid error number.
Source	Name of the current Visual Basic project.
Description	String expression corresponding to the return value of the **Error** function for the specified **Number**, if this string exists. If the string doesn't exist, **Description** contains a zero-length string ("").
HelpFile	The fully qualified drive, path, and file name of the appropriate Visual Basic Help file.
HelpContext	The appropriate Visual Basic Help file context ID for the error corresponding to the **Number** property.
LastDLLError	Zero.

Example:
```
On Error Resume Next    ' Defer error handling.
Error 11    ' Simulate the "Division by zero" error.
```

Declarations

Function Procedures

Declares the name, arguments, and code that form the body of a **Function** procedure.

Syntax

[**Public** | **Private | Friend**] [**Static**] **Function** *name* [**(***arglist***)**] [**As** *type*]
[*statements*]
[*name* = *expression*]
[**Exit Function**]
[*statements*]
[*name* = *expression*]

End Function

The **Function** statement syntax has these parts:

Part	Description
Public	Optional. Indicates that the **Function** procedure is accessible to all other procedures in all modules. If used in a module that contains an **Option Private**, the procedure is not available outside the project.
Private	Optional. Indicates that the **Function** procedure is accessible only to other procedures in the module where it is declared.
Friend	Optional. Used only in a class module. Indicates that the **Function** procedure is visible throughout the project, but not visible to a controller of an instance of an object.
Static	Optional. Indicates that the **Function** procedure's local variables are preserved between calls. The **Static** attribute doesn't affect variables that are declared outside the **Function**, even if they are used in the procedure.
name	Required. Name of the **Function**; follows standard variable naming conventions.
arglist	Optional. List of variables representing arguments that are passed to the **Function** procedure when it is called. Multiple variables are separated by commas.
type	Optional. Data type of the value returned by the **Function** procedure; may be Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal (not currently supported), Date, String, or (except fixed length), Object, Variant, or any user-defined type.
statements	Optional. Any group of statements to be executed within the **Function** procedure.
expression	Optional. Return value of the **Function**.

The *arglist* argument has the following syntax and parts:

[**Optional**] [**ByVal** | **ByRef**] [**ParamArray**] *varname*[**( )**] [**As** *type*] [= *defaultvalue*]

Part	Description
Optional	Optional. Indicates that an argument is not required. If used, all subsequent arguments in *arglist* must also be optional and declared using the **Optional** keyword. **Optional** can't be used for any argument if **ParamArray** is used.
ByVal	Optional. Indicates that the argument is passed by value.
ByRef	Optional. Indicates that the argument is passed by reference. **ByRef** is the default in Visual Basic.
ParamArray	Optional. Used only as the last argument in *arglist* to indicate that the final argument is an **Optional** array of **Variant** elements. The **ParamArray** keyword allows you to provide an arbitrary number of arguments. It may not be used with **ByVal**, **ByRef**, or **Optional**.

varname	Required. Name of the variable representing the argument; follows standard variable naming conventions.
type	Optional. Data type of the argument passed to the procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported) **Date**, **String** (variable length only), **Object**, **Variant**, or a specific object type. If the parameter is not **Optional**, a user-defined type may also be specified.
defaultvalue	Optional. Any constant or constant expression. Valid for **Optional** parameters only. If the type is an **Object**, an explicit default value can only be **Nothing**.

Remarks
If not explicitly specified using **Public**, **Private**, or **Friend**, **Function** procedures are public by default. If **Static** isn't used, the value of local variables is not preserved between calls. The **Friend** keyword can only be used in class modules. However, **Friend** procedures can be accessed by procedures in any module of a project. A **Friend** procedure does't appear in the type library of its parent class, nor can a **Friend** procedure be late bound.

Const

Declares constants for use in place of literal values.
Syntax
[**Public** | **Private**] **Const** *constname* [**As** *type*] = *expression*

The **Const** statement syntax has these parts:

Part	**Description**
Public	Optional. Keyword used at module level to declare constants that are available to all procedures in all modules. Not allowed in procedures.
Private	Optional. Keyword used at module level to declare constants that are available only within the module where the declaration is made. Not allowed in procedures.
constname	Required. Name of the constant; follows standard variable naming conventions.
type	Optional. Data type of the constant; may be Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal (not currently supported), Date, String, or Variant. Use a separate **As** *type* clause for each constant being declared.
expression	Required. Literal, other constant, or any combination that includes all arithmetic or logical operators except **Is**.

Remarks
Constants are private by default. Within procedures, constants are always private; their visibility can't be changed. In standard modules, the default visibility of module-level constants can be changed using the **Public** keyword. In class modules, however, constants can only be private and their visibility can't be changed using the **Public** keyword.

Example:
```
' Constants are Private by default.
Const MyVar = 459
' Declare Public constant.
Public Const MyString = "HELP"
' Declare Private Integer constant.
Private Const MyInt As Integer = 5
' Declare multiple constants on same line.
Const MyStr = "Hello", MyDouble As Double = 3.4567
```

Call Statement

Transfers control to a **Sub** procedure, **Function** procedure, or dynamic-link library (DLL) procedure.

Syntax

[**Call**] *name* [*argumentlist*]

The **Call** statement syntax has these parts:

Part	Description
Call	Optional; keyword. If specified, you must enclose *argumentlist* in parentheses. For example: `Call MyProc(0)`
name	Required. Name of the procedure to call.
argumentlist	Optional. Comma-delimited list of variables, arrays, or expressions to pass to the procedure. Components of *argumentlist* may include the keywords **ByVal** or **ByRef** to describe how the arguments are treated by the called procedure. However, **ByVal** and **ByRef** can be used with **Call** only when calling a DLL procedure. On the Macintosh, **ByVal** and **ByRef** can be used with **Call** when making a call to a Macintosh code resource.

Remarks

You are not required to use the **Call** keyword when calling a procedure. However, if you use the **Call** keyword to call a procedure that requires arguments, *argumentlist* must be enclosed in parentheses.

Example:

```
Call an intrinsic function. The return value of the function is
' discarded.
Call Shell(AppName, 1)    ' AppName contains the path of the
      ' executable file.
```

CallByName

Executes a method of an object, or sets or returns a property of an object.

Syntax

CallByName(*object*, *procname*, *calltype*,*[args()]*)

The **CallByName** function syntax has these named arguments:

Part	Description
object	Required; **Variant** (**Object**). The name of the object on which the function will be executed.
procname	Required; **Variant** (**String**). A string expression containing the name of a property or method of the object.
calltype	Required; **Constant**. A constant of type **vbCallType** representing the type of procedure being called.
args()	Optional: **Variant** (**Array**).

Remarks

The **CallByName** function is used to get or set a property, or invoke a method at run time using a string name.

Example:

```
CallByName Text1, "MousePointer", vbLet, vbCrosshair
Result = CallByName (Text1, "MousePointer", vbGet)
CallByName Text1, "Move", vbMethod, 100, 100
```

Option Explicit

Used at module level to force explicit declaration of all variables in that module.

Syntax

Option Explicit

Remarks

If used, the **Option Explicit** statement must appear in a module before any procedures.
When **Option Explicit** appears in a module, you must explicitly declare all variables using the **Dim**, **Private**, **Public**, **ReDim**, or **Static** statements. If you attempt to use an undeclared variable name, an error occurs at compile time.
If you don't use the **Option Explicit** statement, all undeclared variables are of **Variant** type unless the default type is otherwise specified with a **Def***type* statement.

Example:

```
Option explicit    ' Force explicit variable declaration.
Dim MyVar    ' Declare variable.
MyInt = 10    ' Undeclared variable generates error.
MyVar = 10    ' Declared variable does not generate error.
```

Option Private

When used in host applications that allow references across multiple projects, **Option Private Module** prevents a module's contents from being referenced outside its project. In host applications that don't permit such references, for example, standalone versions of Visual Basic, **Option Private** has no effect.

Syntax

Option Private Module

Remarks

If used, the **Option Private** statement must appear at module level, before any procedures.
When a module contains **Option Private Module**, the public parts, for example, variables, objects, and user-defined types declared at module level, are still available within the project containing the module, but they are not available to other applications or projects.

Example:

```
Option private Module    ' Indicates that module is private.
```

Option Compare

Used at module level to declare the default comparison method to use when string data is compared.

Syntax

Option Compare {**Binary** | **Text** | **Database**}

Remarks

If used, the **Option Compare** statement must appear in a module before any procedures.
The **Option Compare** statement specifies the string comparison method (**Binary**, **Text**, or **Database**) for a module. If a module doesn't include an **Option Compare** statement, the default text comparison method is **Binary**.
Option Compare Binary results in string comparisons based on a sort order derived from the internal binary representations of the characters. In Microsoft Windows, sort order is determined by the code page. A typical binary sort order is shown in the following example:
A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø
Option Compare Text results in string comparisons based on a case-insensitive text sort order determined by your system's locale. When the same characters are sorted using **Option Compare Text**, the following text sort order is produced:
(A=a) < (À=à) < (B=b) < (E=e) < (Ê=ê) < (Z=z) < (Ø=ø)
Example:
Set the string comparison method to Binary.

Option compare Binary ' That is, "AAA" is less than "aaa".
' Set the string comparison method to Text.
Option compare Text ' That is, "AAA" is equal to "aaa".

Type…End Type

Used at module level to define a user-defined data thype containing one or more elements.
Syntax
[**Private** | **Public**] **Type** *varname*
elementname [**(**[*subscripts*]**)**] **As** *type*
[*elementname* [**(**[*subscripts*]**)**] **As** *type*]
. . .
End Type

The **Type** statement syntax has these parts:

Part	Description
Public	Optional. Used to declare user-defined htypes that are available to all procedures in all modules in all projects.
Private	Optional. Used to declare user-defined types that are available only within the module where the declaration is made.
varname	Required. Name of the user-defined type; follows standard variable naming conventions.
elementname	Required. Name of an element of the user-defined type. Element names also follow standard variable naming conventions, except that keywords can be used.
subscripts	When not explicitly stated in *lower*, the lower bound of an array is controlled by the **Option Base** statement. The lower bound is zero if no **Option Base** statement is present.
type	Required. Data type of the element; may be Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal (not currently supported), Date, String (for variable-length strings), **String** * *length* (for fixed-length strings), Object, Variant, another user-defined type, or an object type.

Remarks
The **Type** statement can be used only at module level. Once you have declared a user-defined type using the **Type** statement, you can declare a variable of that type anywhere within the scope of the declaration. Use **Dim**, **Private**, **Public**, **ReDim**, or **Static** to declare a variable of a user-defined type.

Example:

```
Type StateData
   CityCode (1 To 100) As Integer    ' Declare a static array.
   County As String * 30
End Type

Dim Washington(1 To 100) As StateData
```

GetObject

Returns a reference to an object provided by an ActiveX component.
Syntax
GetObject([pathname] [, class])

The GetObject function syntax has these named arguments:

Part Description

pathname	Optional; Variant (String). The full path and name of the file containing the object to retrieve. If pathname is omitted, class is required.
class	Optional; Variant (String). A string representing the class of the object.

The class argument uses the syntax appname.objecttype and has these parts:

Part	Description
appname	Required; Variant (String). The name of the application providing the object.
objecttype	Required; Variant (String). The type or class of object to create.

Remarks
Use the GetObject function to access an ActiveX object from a file and assign the object to an object variable. Use the Set statement to assign the object returned by GetObject to the object variable.

Example:
```
Dim CADObject As Object
Set CADObject = GetObject("C:\CAD\SCHEMA.CAD")
```

CreateObject

Creates and returns a reference to an ActiveX object.
Syntax
CreateObject(*class,[servername]***)**

The **CreateObject** function syntax has these parts:

Part	**Description**
class	Required; **Variant** (**String**). The application name and class of the object to create.
servername	Optional; **Variant** (**String**). The name of the network server where the object will be created. If *servername* is an empty string (""), the local machine is used.

The *class* argument uses the syntax *appname.objecttype* and has these parts:

Part	**Description**
appname	Required; **Variant** (**String**). The name of the application providing the object.
objecttype	Required; **Variant** (**String**). The type or class of object to create.

Remarks
Every application that supports Automation provides at least one type of object. For example, a word processing application may provide an **Application** object, a **Document** object, and a **Toolbar** object.

Example:
```
Dim ExcelSheet As Object
Set ExcelSheet = CreateObject("Excel.Sheet")
```

Let Statement

Assigns the value of an expression to a variable or property.
Syntax
[**Let**] *varname* **=** *expression*

The **Let** statement syntax has these parts:

Part	**Description**

Let	Optional. Explicit use of the **Let** keyword is a matter of style, but it is usually omitted.
varname	Required. Name of the variable or property; follows standard variable naming conventions.
expression	Required. Value assigned to the variable or property.

Remarks
A value expression can be assigned to a variable or property only if it is of a data type that is compatible with the variable. You can't assign string expressions to numeric variables, and you can't assign numeric expressions to string variables. If you do, an error occurs at compile time.
Variant variables can be assigned either string or numeric expressions. However, the reverse is not always true. Any **Variant** except a Null can be assigned to a string variable, but only a **Variant** whose value can be interpreted as a number can be assigned to a numeric variable. Use the **IsNumeric** function to determine if the **Variant** can be converted to a number.

Example:
Dim MyStr, MyInt
' The following variable assignments use the Let statement.
Let MyStr = "Hello World"
Let MyInt = 5

TypeName

Returns a **String** that provides information about a variable.
Syntax
TypeName(*varname***)**

The required *varname* argument is a Variant containing any variable except a variable of a user-defined type.
Remarks
The string returned by **TypeName** can be any one of the following:

String returned	**Variable**
object thype	An object whose type is *objecttype*
Byte	Byte value
Integer	Integer
Long	Long integer
Single	Single-precision floating-point number
Double	Double-precision floating-point number
Currency	Currency value
Decimal	Decimal value
Date	Date value
String	String
Boolean	Boolean value
Error	An error value
Empty	Uninitialized
Null	No valid data

Object	An object
Unknown	An object whose type is unknown
Nothing	Object variable that doesn't refer to an object

If *varname* is an array, the returned string can be any one of the possible returned strings (or **Variant**) with empty parentheses appended. For example, if *varname* is an array of integers, **TypeName** returns "`Integer()`".

Example:
```
Dim MyType
MyType = TypeName(StrVar)    ' Returns "String".
MyType = TypeName(IntVar)    ' Returns "Integer".
MyType = TypeName(CurVar)    ' Returns "Currency".
MyType = TypeName(NullVar)   ' Returns "Null".
MyType = TypeName(ArrayVar)  ' Returns "Integer()".
```

VarType

Returns an **Integer** indicating the subtype of a variable.

Syntax

VarType(*varname***)**

The required *varname* argument is a Variant containing any variable except a variable of a user-defined type.

Return Values

Constant	Value	Description
vbEmpty	0	Empty (uninitialized)
vbNull	1	Null (no valid data)
vbInteger	2	Integer
vbLong	3	Long integer
vbSingle	4	Single-precision floating-point number
vbDouble	5	Double-precision floating-point number
vbCurrency	6	Currency value
vbDate	7	Date value
vbString	8	String
vbObject	9	Object
vbError	10	Error value
vbBoolean	11	Boolean value
vbVariant	12	**Variant** (used only with arrays of variants)
vbDataObject	13	A data access object
vbDecimal	14	Decimal value
vbByte	17	Byte value

vbUserDefinedType	36	Variants that contain user-defined types
vbArray	8192	Array

Note These constants are specified by Visual Basic for Applications. The names can be used anywhere in your code in place of the actual values.

Remarks

The **VarType** function never returns the value for **vbArray** by itself. It is always added to some other value to indicate an array of a particular type. The constant **vbVariant** is only returned in conjunction with **vbArray** to indicate that the argument to the **VarType** function is an array of type **Variant**. For example, the value returned for an array of integers is calculated as **vbInteger** + **vbArray**, or 8194. If an object has a default property, **VarType (***object***)** returns the type of the object's default property.

Example:

```
Dim IntVar, StrVar, DateVar, MyCheck
' Initialize variables.
IntVar = 459: StrVar = "Hello World": DateVar = #2/12/69#
MyCheck = VarType(IntVar)    ' Returns 2.
MyCheck = VarType(DateVar)    ' Returns 7.
MyCheck = VarType(StrVar)    ' Returns 8.
```

DefType

Used at module level to set the default data type for variables, arguments passed to procedures, and the return type for **Function** and **Property Get** procedures whose names start with the specified characters.

Syntax

DefBool *letterrange*[**,** *letterrange*] **. . .**
DefByte *letterrange*[**,** *letterrange*] **. . .**
DefInt *letterrange*[**,** *letterrange*] **. . .**
DefLng *letterrange*[**,** *letterrange*] **. . .**
DefCur *letterrange*[**,** *letterrange*] **. . .**
DefSng *letterrange*[**,** *letterrange*] **. . .**
DefDbl *letterrange*[**,** *letterrange*] **. . .**
DefDec *letterrange*[**,** *letterrange*] **. . .**
DefDate *letterrange*[**,** *letterrange*] **. . .**
DefStr *letterrange*[**,** *letterrange*] **. . .**
DefObj *letterrange*[**,** *letterrange*] **. . .**
DefVar *letterrange*[**,** *letterrange*] **. . .**

The required *letterrange* argument has the following syntax:
letter1[**-***letter2*]
The *letter1* and *letter2* arguments specify the name range for which you can set a default data type. Each argument represents the first letter of the variable, argument, **Function** procedure, or **Property Get** procedure name and can be any letter of the alphabet. The case of letters in *letterrange* isn't significant.

Remarks

The statement name determines the data type:

Statement	Data Type
DefBool	Boolean
DefByte	Byte
DefInt	Integer
DefLng	Long
DefCur	Currency
DefSng	Single

DefDbl	Double
DefDec	Decimal (not currently supported)
DefDate	Date
DefStr	String
DefObj	Object
DefVar	Variant

A **Def***type* statement affects only the module where it is used. For example, a **DefInt** statement in one module affects only the default data type of variables, arguments passed to procedures, and the return type for **Function** and **Property Get** procedures declared in that module; the default data type of variables, arguments, and return types in other modules is unaffected. If not explicitly declared with a **Def***type* statement, the default data type for all variables, all arguments, all **Function** procedures, and all **Property Get** procedures is **Variant**.

Date/Time

Date

Returns a **Variant** (**Date**) containing the current system date.

Syntax

Date

Remarks

To set the system date, use the **Date** statement.

Date, and if the calendar is Gregorian, **Date$** behavior is unchanged by the **Calendar** property setting. If the calendar is Hijri, **Date$** returns a 10-character string of the form *mm-dd-yyyy*, where *mm* (01-12), *dd* (01-30) and *yyyy* (1400-1523) are the Hijri month, day and year. The equivalent Gregorian range is Jan 1, 1980 through Dec 31, 2099.

Example:

```
Dim s as Date
S = Date
Label1 = s
```

Time

Sets the system time.

Syntax

Time = *time*

The required *time* argument is any numeric expression, string expression, or any combination, that can represent a time.

Remarks

If *time* is a string, **Time** attempts to convert it to a time using the time separators you specified for your system. If it can't be converted to a valid time, an error occurs.

Example:

```
Private Sub Form_Load()
Dim s As Date
s = Time
Label1 = s
End Sub
‘Displays the Time in label1
```

Now

Returns a **Variant** (**Date**) specifying the current date and time according your computer's system date and time.

Syntax

Now

Example:

```
Private Sub Form_Load()
Dim s As Date
s = Now
Label1 = s
End Sub
‘Displays the date and time in label1
```

Timer

Timers execute code repeatedly according to the Interval you specify. Set the Interval property in milliseconds. For example, 2000 = 2 seconds. Timers are useful for checking programs conditions periodically, but don't get in the habit of using them for everything. A Timer control is not a clock and should not be relied upon to keep accurate time.

DateAdd

Returns a **Variant** (**Date**) containing a date to which a specified time interval has been added.

Syntax

DateAdd(*interval, number, date*)

The **DateAdd** function syntax has these named arguments:

Part	Description
interval	Required. String expression that is the interval of time you want to add.
number	Required. Numeric expression that is the number of intervals you want to add. It can be positive (to get dates in the future) or negative (to get dates in the past).
date	Required. **Variant** (**Date**) or literal representing date to which the interval is added.

Settings

The ***interval*** argument has these settings:

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

Remarks

You can use the **DateAdd** function to add or subtract a specified time interval from a date. For example, you can use **DateAdd** to calculate a date 30 days from today or a time 45 minutes from now.
To add days to ***date***, you can use Day of Year ("y"), Day ("d"), or Weekday ("w").

Example:

DateAdd("m", 1, "31-Jan-95")
In this case, **DateAdd** returns 28-Feb-95, not 31-Feb-95. If ***date*** is 31-Jan-96, it returns 29-Feb-96 because 1996 is a leap year.

DateDiff

Returns a **Variant** (**Long**) specifying the number of time intervals between two specified dates.

Syntax

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

The **DateDiff** function syntax has these named arguments:

Part	Description
interval	Required. String expression that is the interval of time you use to calculate

	the difference between ***date1*** and ***date2***.
date1, ***date2***	Required; **Variant** (**Date**). Two dates you want to use in the calculation.
firstdayofweek	Optional. A constant that specifies the first day of the week. If not specified, Sunday is assumed.
firstweekofyear	Optional. A constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs.

Settings
The ***interval*** argument has these settings:

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

The ***firstdayofweek*** argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use the NLS API setting.
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Constant	Value	Description
vbUseSystem	0	Use the NLS API setting.
vbFirstJan1	1	Start with week in which January 1 occurs (default).
vbFirstFourDays	2	Start with the first week that has at

		least four days in the new year.
vbFirstFullWeek	3	Start with first full week of the year.

Remarks
You can use the **DateDiff** function to determine how many specified time intervals exist between two dates. For example, you might use **DateDiff** to calculate the number of days between two dates, or the number of weeks between today and the end of the year.

Example:
```
Dim TheDate As Date    ' Declare variables.
Dim Msg
TheDate = InputBox("Enter a date")
Msg = "Days from today: " & DateDiff("d", Now, TheDate)
MsgBox Msg
‘Displays difference between dates in number of days
```

DateSerial
Returns a **Variant** (**Date**) for a specified year, month, and day.
Syntax
DateSerial(*year*, *month*, *day*)

The **DateSerial** function syntax has these named arguments:

Part	**Description**
year	Required; **Integer**. Number between 100 and 9999, inclusive, or a numeric expression.
month	Required; **Integer**. Any numeric expression.
day	Required; **Integer**. Any numeric expression.

Remarks
To specify a date, such as December 31, 1991, the range of numbers for each **DateSerial** argument should be in the accepted range for the unit; that is, 1–31 for days and 1–12 for months. However, you can also specify relative dates for each argument using any numeric expression that represents some number of days, months, or years before or after a certain date.

Example:
```
Dim MyDate
' MyDate contains the date for February 12, 1969.
MyDate = DateSerial(1969, 2, 12)    ' Return a date.
```

DateValue
Returns a **Variant** (**Date**).
Syntax
DateValue(*date*)

The required *date* argument is normally a string expression representing a date from January 1, 100 through December 31, 9999. However, *date* can also be any expression that can represent a date, a time, or both a date and time, in that range.
Remarks
If *date* is a string that includes only numbers separated by valid date separators, **DateValue** recognizes the order for month, day, and year according to the Short Date format you specified for your system. **DateValue** also recognizes unambiguous dates that contain month names, either in long or abbreviated form. For example, in addition to recognizing 12/30/1991 and 12/30/91, **DateValue** also recognizes December 30, 1991 and Dec 30, 1991.

Example:
Dim MyDate
MyDate = **DateValue(**"February 12, 1969"**)** ' Returns 12/02/1965

Year

Returns a **Variant** (**Integer**) containing a whole number representing the year.
Syntax
Year(*date***)**

The required *date* argument is any Variant, numeric expression, string expression, or any combination, that can represent a date. If *date* contains Null, **Null** is returned.

Example:
Dim MyDate, MyYear
MyDate = #February 12, 1969# ' Assign a date.
MyYear = **Year(**MyDate**)** ' MyYear contains 1969.

Month

Returns a **Variant** (**Integer**) specifying a whole number between 1 and 12, inclusive, representing the month of the year.
Syntax
Month(*date***)**

The required *date* argument is any Variant, numeric expression, string expression, or any combination, that can represent a date. If *date* contains Null, **Null** is returned.

Example:
Dim MyDate, MyMonth
MyDate = #February 12, 1969# ' Assign a date.
MyMonth = **Month(**MyDate**)** ' MyMonth contains 2.

MonthName

Returns a string indicating the specified month.
Syntax
MonthName(*month*[**,** *abbreviate*]**)**

The **MonthName** function syntax has these parts:

Part	Description
month	Required. The numeric designation of the month. For example, January is 1, February is 2, and so on.
abbreviate	Optional. **Boolean** value that indicates if the month name is to be abbreviated. If omitted, the default is **False**, which means that the month name is not abbreviated.

Example:
Private Sub Form_Load()
Label1 = **MonthName**(11)
End Sub
'Returns November

WeekDayName

Returns a string indicating the specified day of the week.
Syntax
WeekdayName(*weekday***,** *abbreviate***,** *firstdayofweek***)**

The **WeekdayName** function syntax has these parts:

Part	Description
weekday	Required. The numeric designation for the day of the week. Numeric value of each day depends on setting of the *firstdayofweek* setting.
abbreviate	Optional. **Boolean** value that indicates if the weekday name is to be abbreviated. If omitted, the default is **False**, which means that the weekday name is not abbreviated.
firstdayofweek	Optional. Numeric value indicating the first day of the week. See Settings section for values.

Settings
The *firstdayofweek* argument can have the following values:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Example:
```
Label1 = WeekdayName(3)
‘Returns Wednesday
```

Day

Returns a Variant (Integer) specifying a whole number between 1 and 31, inclusive, representing the day of the month.

Syntax

Day(date)

The required date argument is any Variant, numeric expression, string expression, or any combination, that can represent a date. If date contains Null, Null is returned.

Example:
```
Dim MyDate, MyDay
MyDate = #February 12, 1969#    ' Assign a date.
MyDay = Day(MyDate)    ' MyDay contains 12.
```

Hour

Returns a **Variant** (**Integer**) specifying a whole number between 0 and 23, inclusive, representing the hour of the day.

Syntax

Hour(*time***)**

The required *time* argument is any Variant, numeric expression, string expression, or any combination, that can represent a time. If *time* contains Null, **Null** is returned.

Example:
Dim MyTime, MyHour
MyTime = #4:35:17 PM# ' Assign a time.
MyHour = **Hour(**MyTime**)** ' MyHour contains 16.

Minute

Returns a **Variant** (**Integer**) specifying a whole number between 0 and 59, inclusive, representing the minute of the hour.
Syntax
Minute(*time***)**

The required *time* argument is any Variant, numeric expression, string expression, or any combination, that can represent a time. If *time* contains Null, **Null** is returned.

Example:
Dim MyTime, MyMinute
MyTime = #4:35:17 PM# ' Assign a time.
MyMinute = **Minute**(MyTime) ' MyMinute contains 35.

Second

Returns a **Variant** (**Integer**) specifying a whole number between 0 and 59, inclusive, representing the second of the minute.
Syntax
Second(*time***)**

The required *time* argument is any Variant, numeric expression, string expression, or any combination, that can represent a time. If *time* contains Null, **Null** is returned.

Example:
Dim MyTime, MySecond
MyTime = #4:35:17 PM# ' Assign a time.
MySecond = **Second(**MyTime**)** ' MySecond contains 17.

TimeSerial

Returns a **Variant** (**Date**) containing the time for a specific hour, minute, and second.
Syntax
TimeSerial(*hour, minute, second*)

The **TimeSerial** function syntax has these named arguments:

Part	**Description**
hour	Required; **Variant** (**Integer**). Number between 0 (12:00 A.M.) and 23 (11:00 P.M.), inclusive, or a numeric expression.
minute	Required; **Variant** (**Integer**). Any numeric expression.
second	Required; **Variant** (**Integer**). Any numeric expression.

Remarks
To specify a time, such as 11:59:59, the range of numbers for each **TimeSerial** argument should be in the normal range for the unit; that is, 0–23 for hours and 0–59 for minutes and seconds. However, you can also specify relative times for each argument using any numeric expression that represents some number of hours, minutes, or seconds before or after a certain time. The following example uses expressions instead of absolute time numbers.

Example:
Dim MyTime
MyTime = **TimeSerial(**16**,** 35**,** 17**)** ' MyTime contains serial
  ' representation of 4:35:17 PM.

TimeValue

Returns a **Variant** (**Date**) containing the time.
Syntax
TimeValue(*time***)**

The required *time* argument is normally a string expression representing a time from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.), inclusive. However, *time* can also be any expression that represents a time in that range. If *time* contains Null, **Null** is returned.

Example:

```
Dim MyTime
MyTime = TimeValue("4:35:17 PM")    ' Returns 4:35:17 PM
```

WeekDay

Returns a **Variant** (**Integer**) containing a whole number representing the day of the week.
Syntax
Weekday(*date*, [***firstdayofweek***])**

The **Weekday** function syntax has these named arguments:

Part	Description
date	Required. Variant, numeric expression, string expression, or any combination, that can represent a date. If ***date*** contains Null, **Null** is returned.
firstdayofweek	Optional. A constant that specifies the first day of the week. If not specified, **vbSunday** is assumed.

Settings
The ***firstdayofweek*** argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use the NLS API setting.
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Return Values
The **Weekday** function can return any of these values:

Constant	Value	Description
vbSunday	1	Sunday

vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Example:

```
Dim MyDate, MyWeekDay
MyDate = #February 12, 1969#    ' Assign a date.
MyWeekDay = Weekday(MyDate)    ' MyWeekDay contains 4 because
   ' MyDate represents a Wednesday.
```

Miscellaneous

MsgBox

Displays a message in a dialog box, waits for the user to click a button, and returns an **Integer** indicating which button the user clicked.

Syntax

MsgBox(*prompt*[, *buttons*] [, *title*] [, *helpfile*, *context*])

The **MsgBox** function syntax has these named arguments:

Part	Description
prompt	Required. String expression displayed as the message in the dialog box. The maximum length of ***prompt*** is approximately 1024 characters, depending on the width of the characters used. If ***prompt*** consists of more than one line, you can separate the lines using a carriage return character (**Chr(**13**)**), a linefeed character (**Chr(**10**)**), or carriage return – linefeed character combination (**Chr(**13**)** & **Chr(**10**)**) between each line.
buttons	Optional. Numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If omitted, the default value for ***buttons*** is 0.
title	Optional. String expression displayed in the title bar of the dialog box. If you omit ***title***, the application name is placed in the title bar.
helpfile	Optional. String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If ***helpfile*** is provided, ***context*** must also be provided.
context	Optional. Numeric expression that is the Help context number assigned to the appropriate Help topic by the Help author. If ***context*** is provided, ***helpfile*** must also be provided.

Settings

The ***buttons*** argument settings are:

Constant	Value	Description
vbOKOnly	0	Display **OK** button only.
vbOKCancel	1	Display **OK** and **Cancel** buttons.
vbAbortRetryIgnore	2	Display **Abort**, **Retry**, and **Ignore** buttons.
vbYesNoCancel	3	Display **Yes**, **No**, and **Cancel** buttons.
vbYesNo	4	Display **Yes** and **No** buttons.
vbRetryCancel	5	Display **Retry** and **Cancel** buttons.
vbCritical	16	Display **Critical Message** icon.
vbQuestion	32	Display **Warning Query** icon.
vbExclamation	48	Display **Warning Message** icon.
vbInformation	64	Display **Information Message** icon.
vbDefaultButton1	0	First button is default.
vbDefaultButton2	256	Second button is default.
vbDefaultButton3	512	Third button is default.

vbDefaultButton4	768	Fourth button is default.
vbApplicationModal	0	Application modal; the user must respond to the message box before continuing work in the current application.
vbSystemModal	4096	System modal; all applications are suspended until the user responds to the message box.
vbMsgBoxHelpButton	16384	Adds Help button to the message box
VbMsgBoxSetForeground	65536	Specifies the message box window as the foreground window
vbMsgBoxRight	524288	Text is right aligned
vbMsgBoxRtlReading	1048576	Specifies text should appear as right-to-left reading on Hebrew and Arabic systems

Shell

Runs an executable program and returns a **Variant** (**Double**) representing the program's task ID if successful, otherwise it returns zero.

Syntax

Shell(*pathname*[,*windowstyle*])

The **Shell** function syntax has these named arguments:

Part	Description
pathname	Required; **Variant** (**String**). Name of the program to execute and any required arguments or command-line switches; may include directory or folder and drive.
windowstyle	Optional. **Variant** (**Integer**) corresponding to the style of the window in which the program is to be run. If ***windowstyle*** is omitted, the program is started minimized with focus. On the Macintosh (System 7.0 or later), ***windowstyle*** only determines whether or not the application gets the focus when it is run.

The ***windowstyle*** named argument has these values:

Constant	Value	Description
vbHide	0	Window is hidden and focus is passed to the hidden window. The **vbHide** constant is not applicable on Macintosh platforms.
vbNormalFocus	1	Window has focus and is restored to its original size and position.
vbMinimizedFocus	2	Window is displayed as an icon with focus.
vbMaximizedFocus	3	Window is maximized with focus.
vbNormalNoFocus	4	Window is restored to its most recent size and position. The currently active window remains active.
vbMinimizedNoFocus	6	Window is displayed as an icon. The currently active window remains active.

Remarks
If the **Shell** function successfully executes the named file, it returns the task ID of the started program. The task ID is a unique number that identifies the running program. If the **Shell** function can't start the named program, an error occurs.

Example:
```
' Specifying 1 as the second argument opens the application in
' normal size and gives it the focus.
Dim RetVal
RetVal = Shell("C:\WINDOWS\CALC.EXE", 1)    ' Run Calculator.
```

RGB
Returns a Long whole number representing an RGB color value.
Syntax
RGB(*red*, *green*, *blue*)

The **RGB** function syntax has these named arguments:

Part	Description
red	Required; **Variant** (**Integer**). Number in the range 0–255, inclusive, that represents the red component of the color.
green	Required; **Variant** (**Integer**). Number in the range 0–255, inclusive, that represents the green component of the color.
blue	Required; **Variant** (**Integer**). Number in the range 0–255, inclusive, that represents the blue component of the color.

Remarks
Application methods and properties that accept a color specification expect that specification to be a number representing an RGB color value. An RGB color value specifies the relative intensity of red, green, and blue to cause a specific color to be displayed.
The value for any argument to **RGB** that exceeds 255 is assumed to be 255.
The following table lists some standard colors and the red, green, and blue values they include:

Color	Red Value	Green Value	Blue Value
Black	0	0	0
Blue	0	0	255
Green	0	255	0
Cyan	0	255	255
Red	255	0	0
Magenta	255	0	255
Yellow	255	255	0
White	255	255	255

Example:
```
Dim RED, I, RGBValue, MyObject
Red = RGB(255, 0, 0)    ' Return the value for Red.
I = 75    ' Initialize offset.
RGBValue = RGB(I, 64 + I, 128 + I)    ' Same as RGB(75, 139, 203).
MyObject.Color = RGB(255, 0, 0)    ' Set the Color property of
   ' MyObject to Red.
```

QBColor

Returns a Long representing the RGB color code corresponding to the specified color number.

Syntax

QBColor(*color***)**

The required *color* argument is a whole number in the range 0–15.

Settings

The *color* argument has these settings:

Number	Color	Number	Color
0	Black	8	Gray
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Magenta
6	Yellow	14	Light Yellow
7	White	15	Bright White

Example:

```
Sub ChangeBackColor (ColorCode As Integer, MyForm As Form)
   MyForm.BackColor = QBColor(ColorCode)
End Sub
```

Beep

Sounds a tone through the computer's speaker.

Syntax

Beep

Remarks

The frequency and duration of the beep depend on your hardware and system software, and vary among computers.

Example:

```
Dim I
For I = 1 To 3    ' Loop 3 times.
   Beep    ' Sound a tone.
Next I
```

InputBox

Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns a String containing the contents of the text box.

Syntax

InputBox(*prompt*[, *title*] [, *default*] [, *xpos*] [, *ypos*] [, *helpfile*, *context*])

The **InputBox** function syntax has these named arguments:

Part	Description
prompt	Required. String expression displayed as the message in the dialog box. The maximum length of ***prompt*** is approximately 1024 characters, depending on the width of the characters used. If ***prompt*** consists of more than one line, you can separate the lines using a carriage return character (**Chr(**13**)**), a linefeed character (**Chr(**10**)**), or carriage return–linefeed character combination (**Chr(**13**)** & **Chr(**10**)**) between each line.
title	Optional. String expression displayed in the title bar of the dialog box. If you omit ***title***, the application name is placed in the title bar.
default	Optional. String expression displayed in the text box as the default response if no other input is provided. If you omit ***default***, the text box is displayed empty.
xpos	Optional. Numeric expression that specifies, in twips, the horizontal distance of the left edge of the dialog box from the left edge of the screen. If ***xpos*** is omitted, the dialog box is horizontally centered.
ypos	Optional. Numeric expression that specifies, in twips, the vertical distance of the upper edge of the dialog box from the top of the screen. If ***ypos*** is omitted, the dialog box is vertically positioned approximately one-third of the way down the screen.
helpfile	Optional. String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If ***helpfile*** is provided, ***context*** must also be provided.
context	Optional. Numeric expression that is the Help context number assigned to the appropriate Help topic by the Help author. If ***context*** is provided, ***helpfile*** must also be provided.

Example:

```
Dim Message, Title, Default, MyValue
Message = "Enter a value between 1 and 3"    ' Set prompt.
Title = "InputBox Demo"    ' Set title.
Default = "1"    ' Set default.
' Display message, title, and default value.
MyValue = InputBox(Message, Title, Default)

' Use Helpfile and context. The Help button is added automatically.
MyValue = InputBox(Message, Title, , , , "DEMO.HLP", 10)

' Display dialog box at position 100, 100.
MyValue = InputBox(Message, Title, Default, 100, 100)
```

Load

Loads an object but doesn't show it.

Syntax

Load *object*

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

When an object is loaded, it is placed in memory, but isn't visible. Use the **Show** method to make the object visible. Until an object is visible, a user can't interact with it. The object can be manipulated programmatically in its Initialize event procedure.

Example:

```
Private Sub Command1_Click ()
   Load Form2
   Form2.Show
End Sub
```

UnLoad

Removes an object from memory.

Syntax

Unload *object*

The required *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

When an object is unloaded, it's removed from memory and all memory associated with the object is reclaimed. Until it is placed in memory again using the **Load** statement, a user can't interact with an object, and the object can't be manipulated programmatically.

Example:

```
Private Sub Command2_Click()
   Form2.Hide
   Unload Form2
End Sub
```

SendKeys

Sends one or more keystrokes to the active window as if typed at the keyboard.

Syntax

SendKeys ***string***[, ***wait***]

The **SendKeys** statement syntax has these named arguments:

Part	**Description**
string	Required. String expression specifying the keystrokes to send.
Wait	Optional. Boolean value specifying the wait mode. If **False** (default), control is returned to the procedure immediately after the keys are sent. If **True**, keystrokes must be processed before control is returned to the procedure.

Remarks

Each key is represented by one or more characters. To specify a single keyboard character, use the character itself. For example, to represent the letter A, use "A" for ***string***. To represent more than one character, append each additional character to the one preceding it. To represent the letters A, B, and C, use "ABC" for ***string***.
The plus sign (**+**), caret (^), percent sign (**%**), tilde (~), and parentheses **( )** have special meanings to **SendKeys**. To specify one of these characters, enclose it within braces ({}). For example, to specify the plus sign, use {+}. Brackets ([ ]) have no special meaning to **SendKeys**, but you must enclose them in braces. In other applications, brackets do have a special meaning that may be significant when dynamic data exchange (DDE) occurs. To specify brace characters, use {{} and {}}.
To specify characters that aren't displayed when you press a key, such as ENTER or TAB, and keys that represent actions rather than characters, use the codes shown below:

KeyCodes

Key	Code
BACKSPACE	{BACKSPACE}, {BS}, or {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL or DELETE	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER}or ~
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS or INSERT	{INSERT} or {INS}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC}
RIGHT ARROW	{RIGHT}

Key	Code
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}

To specify keys combined with any combination of the SHIFT, CTRL, and ALT keys, precede the key code with one or more of the following codes:

Key	Code
SHIFT	+
CTRL	^
ALT	%

To specify that any combination of SHIFT, CTRL, and ALT should be held down while several other keys are pressed, enclose the code for those keys in parentheses. For example, to specify to hold down SHIFT while E and C are pressed, use "+(EC)". To specify to hold down SHIFT while E is pressed, followed by C without SHIFT, use "+EC".
To specify repeating keys, use the form {key number}. You must put a space between key and number. For example, {LEFT 42} means press the LEFT ARROW key 42 times; {h 10} means press H 10 times.

Example:
```
Command1_Click ()
        Text1.SetFocus
SendKeys “{Backspace}”
End Sub
‘Deletes last character in Text1

Command1_Click ()
        SendKeys "%{F4}"
End Sub
‘Closes current window/program
```

LoadPicture

Specifies the bitmap to display on an object.

Syntax

object.**Picture** = **LoadPicture(** *pathname* **)**

The **Picture** property syntax has these parts:

Part	Description
object	Required. A valid object.
pathname	Required. The full path to a picture file.

Example:
```
Command1_Click ()
        Image1.Picture = LoadPicture(C:\Blah.jpg)
End Sub
‘Loads the Picture Blah.jpg in Image1
```

AppActivate

Activates an application window.

Syntax

AppActivate ***title***[, ***wait***]

The **AppActivate** statement syntax has these named arguments:

Part	Description
title	Required. String expression specifying the title in the title bar of the application window you want to activate. The task ID returned by the **Shell** function can be used in place of ***title*** to activate an application.
wait	Optional. Boolean value specifying whether the calling application has the focus before activating another. If **False** (default), the specified application is immediately activated, even if the calling application does not have the focus. If **True**, the calling application waits until it has the focus, then activates the specified application.

Remarks

The **AppActivate** statement changes the focus to the named application or window but does not affect whether it is maximized or minimized. Focus moves from the activated application window when the user takes some action to change the focus or close the window. Use the **Shell** function to start an application and set the window style.

Example:
```
Command1_Click ()
        AppActivate "Microsoft Word"    ' Activates Microsoft Word.
End Sub
```

Values

IsNull

Returns a **Boolean** value that indicates whether an expression contains no valid data (Null).

Syntax

IsNull(*expression***)**

The required *expression* argument is a Variant containing a numeric expression or string expression.

Remarks

IsNull returns **True** if *expression* is **Null**; otherwise, **IsNull** returns **False**. If *expression* consists of more than one variable, **Null** in any constituent variable causes **True** to be returned for the entire expression

Example:

```
MyVar = ""
MyCheck = IsNull(MyVar)    ' Returns False.

MyVar = Null
MyCheck = IsNull(MyVar)    ' Returns True.
```

IsEmpty

Returns a **Boolean** value indicating whether a variable has been initialized.

Syntax

IsEmpty(*expression***)**

The required *expression* argument is a Variant containing a numeric or string expression. However, because **IsEmpty** is used to determine if individual variables are initialized, the *expression* argument is most often a single variable name.

Remarks

IsEmpty returns **True** if the variable is uninitialized, or is explicitly set to Empty; otherwise, it returns **False**. **False** is always returned if *expression* contains more than one variable. **IsEmpty** only returns meaningful information for variants.

Example:

```
MyVar = Null    ' Assign Null.
MyCheck = IsEmpty(MyVar)    ' Returns False.

MyVar = Empty    ' Assign Empty.
MyCheck = IsEmpty(MyVar)    ' Returns True.
```

IsNumeric

Returns a **Boolean** value indicating whether an expression can be evaluated as a number.

Syntax

IsNumeric(*expression***)**

The required *expression* argument is a Variant containing a numeric expression or string expression.

Remarks

IsNumeric returns **True** if the entire *expression* is recognized as a number; otherwise, it returns **False**. **IsNumeric** returns **False** if *expression* is a date expression.

Example:

```
MyVar = "459.95"    ' Assign value.
MyCheck = IsNumeric(MyVar)    ' Returns True.

MyVar = "45 Help"    ' Assign value.
MyCheck = IsNumeric(MyVar)    ' Returns False.
```

Loops and Conditional

If...Then...Else Statement

Conditionally executes a group of statements, depending on the value of an expression.

Syntax

If *condition* **Then** [*statements*] [**Else** *elsestatements*]
Or, you can use the block form syntax:
If *condition* **Then**
[*statements*]
[**ElseIf** *condition-n* **Then**
[*elseifstatements*] ...
[**Else**
[*elsestatements*]]
End If

The **If...Then...Else** statement syntax has these parts:

Part	Description
condition	Required. One or more of the following two types of expressions: A numeric expression or string expression that evaluates to **True** or **False**. If *condition* is Null, *condition* is treated as **False**. An expression of the form **TypeOf** *objectname* **Is** *objecttype*. The *objectname* is any object reference and *objecttype* is any valid object type. The expression is **True** if *objectname* is of the object type specified by *objecttype*; otherwise it is **False**.
statements	Optional in block form; required in single-line form that has no **Else** clause. One or more statements separated by colons; executed if *condition* is **True**.
condition-n	Optional. Same as *condition*.
elseifstatements	Optional. One or more statements executed if associated *condition-n* is **True**.
elsestatements	Optional. One or more statements executed if no previous *condition* or *condition-n* expression is **True**.

Example:

```
Dim Number, Digits, MyString
Number = 53    ' Initialize variable.
If Number < 10 Then
   Digits = 1
ElseIf Number < 100 Then
' Condition evaluates to True so the next statement is executed.
   Digits = 2
Else
   Digits = 3
End If
```

End Statements

Ends a procedure or block.

Syntax

End
End Function
End If
End Property
End Select
End Sub

End Type
End With

The **End** statement syntax has these forms:

Statement	Description
End	Terminates execution immediately. Never required by itself but may be placed anywhere in a procedure to end code execution, close files opened with the **Open** statement and to clear variables.
End Function	Required to end a **Function** statement.
End If	Required to end a block **If…Then…Else** statement.
End Property	Required to end a **Property Let**, **Property Get**, or **Property Set** procedure.
End Select	Required to end a **Select Case** statement.
End Sub	Required to end a **Sub** statement.
End Type	Required to end a user-defined type definition (**Type** statement).
End With	Required to end a **With** statement.

Remarks
When executed, the **End** statement resets all module-level variables and all static local variables in all modules. To preserve the value of these variables, use the **Stop** statement instead. You can then resume execution while preserving the value of those variables.

Example:
```
Sub Form_Load
   Dim Password, Pword
   PassWord = "Swordfish"
   Pword = InputBox("Type in your password")
   If Pword <> PassWord Then
      MsgBox "Sorry, incorrect password"
      End
   End If
End Sub
```

Stop
Suspends execution.
Syntax
Stop

Remarks
You can place **Stop** statements anywhere in procedures to suspend execution. Using the **Stop** statement is similar to setting a breahkpoint in the code.
The **Stop** statement suspends execution, but unlike **End**, it doesn't close any files or clear variables, unless it is in a compiled executable (.exe) file.

Example:
```
If Label1 = “Blah” then
   Stop
End if
```

Switch

Evaluates a list of expressions and returns a **Variant** value or an expression associated with the first expression in the list that is **True**.

Syntax

Switch(*expr-1***,** *value-1*[**,** *expr-2***,** *value-2* **…** [**,** *expr-n***,***value-n*]]**)**

The **Switch** function syntax has these parts:

Part	**Description**
expr	Required. Variant expression you want to evaluate.
value	Required. Value or expression to be returned if the corresponding expression is **True**.

Remarks

The **Switch** function argument list consists of pairs of expressions and values. The expressions are evaluated from left to right, and the value associated with the first expression to evaluate to **True** is returned. If the parts aren't properly paired, a run-time error occurs. For example, if *expr-1* is **True**, **Switch** returns *value-1*. If *expr-1* is **False**, but *expr-2* is **True**, **Switch** returns *value-2*, and so on.

Switch returns a Null value if:

- None of the expressions is **True**.
- The first **True** expression has a corresponding value that is **Null**.

Example:

```
Function MatchUp (CityName As String)
   Matchup = Switch(CityName = "London", "English", CityName _
             = "Rome", "Italian", CityName = "Paris", "French")
End Function
'This example uses the Switch function to return the name of a language that matches the name of a city.
```

Goto

Branches unconditionally to a specified line within a procedure.

Syntax

GoTo *line*

The required *line* argument can be any line label or line number.

Remarks

GoTo can branch only to lines within the procedure where it appears.

Note Too many **GoTo** statements can make code difficult to read and debug. Use structured control statements (**Do...Loop**, **For...Next**, **If...Then...Else**, **Select Case**) whenever possible.

Example:

```
If Label1 = "Blah" then
        Goto Something
Else
        End
Something:
End if
```

On...GoSub, On...GoTo Statements

Branch to one of several specified lines, depending on the value of an expression.

Syntax

On *expression* **GoSub** *destinationlist*
On *expression* **GoTo** *destinationlist*

The **On...GoSub** and **On...GoTo** statement syntax has these parts:

Part	Description
expression	Required. Any numeric expression that evaluates to a whole number between 0 and 255, inclusive. If *expression* is any number other than a whole number, it is rounded before it is evaluated.
destinationlist	Required. List of line numbers or line labels separated by commas.

Remarks

The value of *expression* determines which line is branched to in *destinationlist*. If the value of *expression* is less than 1 or greater than the number of items in the list, one of the following results occurs:

If *expression* is	Then
Equal to 0	Control drops to the statement following **On...GoSub** or **On...GoTo**.
Greater than number of items in list	Control drops to the statement following **On...GoSub** or **On...GoTo**.
Negative	An error occurs.
Greater than 255	An error occurs.

You can mix line numbers and line labels in the same list. You can use as many line labels and line numbers as you like with **On...GoSub** and **On...GoTo**. However, if you use more labels or numbers than fit on a single line, you must use the line-continuation character to continue the logical line onto the next physical line.

GoSub...Return Statement

Branches to and returns from a subroutine within a procedure.

Syntax

GoSub *line*
. . .
line
. . .
Return

The *line* argument can be any line label or line number.

Remarks

You can use **GoSub** and **Return** anywhere in a procedure, but **GoSub** and the corresponding **Return** statement must be in the same procedure. A subroutine can contain more than one **Return** statement, but the first **Return** statement encountered causes the flow of execution to branch back to the statement immediately following the most recently executed **GoSub** statement.

Note You can't enter or exit **Sub** procedures with **GoSub...Return**.

Example:
```
Sub GosubDemo()
Dim Num
' Solicit a number from the user.
   Num = InputBox("Enter a positive number to be divided by 2.")
' Only use routine if user enters a positive number.
   If Num > 0 Then GoSub MyRoutine
   Debug.Print Num
   Exit Sub    ' Use Exit to prevent an error.
MyRoutine:
   Num = Num/2    ' Perform the division.
   Return    ' Return control to statement.
End Sub    ' following the GoSub statement.
```

With Statement

Executes a series of statements on a single object or a user-defined type.

Syntax

With *object*
[*statements*]
End With

The **With** statement syntax has these parts:

Part	Description
object	Required. Name of an object or a user-defined type.
statements	Optional. One or more statements to be executed on *object*.

Remarks

The **With** statement allows you to perform a series of statements on a specified object without requalifying the name of the object. For example, to change a number of different properties on a single object, place the property assignment statements within the **With** control structure, referring to the object once instead of referring to it with each property assignment. The following example illustrates use of the **With** statement to assign values to several properties of the same object.

Example:
```
With MyLabel
   .Height = 2000
   .Width = 2000
   .Caption = "This is MyLabel"
End With
```

For...Next Statement

Repeats a group of statements a specified number of times.

Syntax

For *counter* **=** *start* **To** *end* [**Step** *step*]
[*statements*]
[**Exit For**]
[*statements*]
Next [*counter*]

The **For…Next** statement syntax has these parts:

Part	Description
counter	Required. Numeric variable used as a loop counter. The variable can't be a Boolean or an array element.
start	Required. Initial value of *counter*.

end	Required. Final value of *counter*.
step	Optional. Amount *counter* is changed each time through the loop. If not specified, *step* defaults to one.
statements	Optional. One or more statements between **For** and **Next** that are executed the specified number of times.

Remarks
The *step* argument can be either positive or negative. The value of the *step* argument determines loop processing as follows:

Value	**Loop executes if**
Positive or 0	*counter* <= *end*
Negative	*counter* >= *end*

After all statements in the loop have executed, *step* is added to *counter*. At this point, either the statements in the loop execute again (based on the same test that caused the loop to execute initially), or the loop is exited and execution continues with the statement following the **Next** statement.

Example:
```
Dim Words, Chars, MyString
For Words = 10 To 1 Step -1     ' Set up 10 repetitions.
   For Chars = 0 To 9    ' Set up 10 repetitions.
      MyString = MyString & Chars    ' Append number to string.
   Next Chars    ' Increment counter
   MyString = MyString & " "    ' Append a space.
Next Words
```

While...Wend Statement

Executes a series of statements as long as a given condition is **True**.

Syntax

While *condition*
[*statements*]
Wend

The **While...Wend** statement syntax has these parts:

Part	**Description**
condition	Required. Numeric expression or string expression that evaluates to **True** or **False**. If *condition* is Null, *condition* is treated as **False**.
statements	Optional. One or more statements executed while condition is **True**.

Remarks
If *condition* is **True**, all *statements* are executed until the **Wend** statement is encountered. Control then returns to the **While** statement and *condition* is again checked. If *condition* is still **True**, the process is repeated. If it is not **True**, execution resumes with the statement following the **Wend** statement.
While...Wend loops can be nested to any level. Each **Wend** matches the most recent **While**.
Tip The **Do...Loop** statement provides a more structured and flexible way to perform looping.

Example:
```
Dim Counter
Counter = 0    ' Initialize variable.
While Counter < 20    ' Test value of Counter.
   Counter = Counter + 1    ' Increment Counter.
Wend    ' End While loop when Counter > 19.
Debug.Print Counter    ' Prints 20 in the Immediate window.
```

Do...Loop Statement

Repeats a block of statements while a condition is **True** or until a condition becomes **True**.

Syntax

Do [{**While** | **Until**} *condition*]
[*statements*]
[**Exit Do**]
[*statements*]
Loop

Or, you can use this syntax:

Do
[*statements*]
[**Exit Do**]
[*statements*]
Loop [{**While** | **Until**} *condition*]

The **Do Loop** statement syntax has these parts:

Part	Description
condition	Optional. Numeric expression or string expression that is **True** or **False**. If *condition* is Null, *condition* is treated as **False**.
statements	One or more statements that are repeated while, or until, *condition* is **True**.

Remarks

Any number of **Exit Do** statements may be placed anywhere in the **Do…Loop** as an alternate way to exit a **Do…Loop**. **Exit Do** is often used after evaluating some condition, for example, **If…Then**, in which case the **Exit Do** statement transfers control to the statement immediately following the **Loop**. When used within nested **Do…Loop** statements, **Exit Do** transfers control to the loop that is one nested level above the loop where **Exit Do** occurs.

Example:

```
Dim Check, Counter
Check = True: Counter = 0    ' Initialize variables.
Do    ' Outer loop.
   Do While Counter < 20    ' Inner loop.
      Counter = Counter + 1    ' Increment Counter.
      If Counter = 10 Then    ' If condition is True.
         Check = False    ' Set value of flag to False.
         Exit Do    ' Exit inner loop.
      End If
   Loop
Loop Until Check = False    ' Exit outer loop immediately.
```

IIF

Returns one of two parts, depending on the evaluation of an expression.

Syntax

IIf(*expr, truepart, falsepart*)

The **IIf** function syntax has these named arguments:

Part	Description
expr	Required. Expression you want to evaluate.
truepart	Required. Value or expression returned if ***expr*** is **True**.
falsepart	Required. Value or expression returned if ***expr*** is **False**.

Remarks
IIf always evaluates both ***truepart*** and ***falsepart***, even though it returns only one of them. Because of this, you should watch for undesirable side effects. For example, if evaluating ***falsepart*** results in a division by zero error, an error occurs even if ***expr*** is **True**.

Example:
```
Function CheckIt (TestMe As Integer)
   CheckIt = IIf(TestMe > 1000, "Large", "Small")
End Function
```

For Each...Next Statement

Repeats a group of statements for each element in an array or collection.
Syntax
For Each *element* **In** *group*
[*statements*]
[**Exit For**]
[*statements*]
Next [*element*]

The **For...Each...Next** statement syntax has these parts:

Part	Description
element	Required. Variable used to iterate through the elements of the collection or array. For collections, *element* can only be a Variant variable, a generic object variable, or any specific object variable. For arrays, *element* can only be a **Variant** variable.
group	Required. Name of an object collection or array (except an array of user-defined types).
statements	Optional. One or more statements that are executed on each item in *group*.

Remarks
The **For...Each** block is entered if there is at least one element in *group*. Once the loop has been entered, all the statements in the loop are executed for the first element in *group*. If there are more elements in *group*, the statements in the loop continue to execute for each element. When there are no more elements in *group*, the loop is exited and execution continues with the statement following the **Next** statement.
Any number of **Exit For** statements may be placed anywhere in the loop as an alternative way to exit. **Exit For** is often used after evaluating some condition, for example **If…Then**, and transfers control to the statement immediately following **Next**.

Example:
```
Dim Found, MyObject, MyCollection
Found = False    ' Initialize variable.
For Each MyObject In MyCollection    ' Iterate through each element.
   If MyObject.Text = "Hello" Then    ' If Text equals "Hello".
      Found = True    ' Set Found to True.
      Exit For    ' Exit loop.
   End If
Next
```

Select Case Statement

Executes one of several groups of statements, depending on the value of an expression.

Syntax

Select Case *testexpression*
[**Case** *expressionlist-n*
[*statements-n*]] ...
[**Case Else**
[*elsestatements*]]
End Select

The **Select Case** statement syntax has these parts:

Part	Description
testexpression	Required. Any numeric expression or string expression.
expressionlist-n	Required if a **Case** appears. Delimited list of one or more of the following forms: *expression*, *expression* **To** *expression*, **Is** *comparisonoperator expression*. The **To** keyword specifies a range of values. If you use the **To** keyword, the smaller value must appear before **To**. Use the **Is** keyword with comparison operators (except **Is** and **Like**) to specify a range of values. If not supplied, the **Is** keyword is automatically inserted.
statements-n	Optional. One or more statements executed if *testexpression* matches any part of *expressionlist-n.*
elsestatements	Optional. One or more statements executed if *testexpression* doesn't match any of the **Case** clause.

Remarks

If *testexpression* matches any **Case** *expressionlist* expression, the *statements* following that **Case** clause are executed up to the next **Case** clause, or, for the last clause, up to **End Select**. Control then passes to the statement following **End Select**. If *testexpression* matches an *expressionlist* expression in more than one **Case** clause, only the statements following the first match are executed.

The **Case Else** clause is used to indicate the *elsestatements* to be executed if no match is found between the *testexpression* and an *expressionlist* in any of the other **Case** selections. Although not required, it is a good idea to have a **Case Else** statement in your **Select Case** block to handle unforeseen *testexpression* values. If no **Case** *expressionlist* matches *testexpression* and there is no **Case Else** statement, execution continues at the statement following **End Select**.

Example:

```
Dim Number
Number = 8    ' Initialize variable.
Select Case Number    ' Evaluate Number.
Case 1 To 5    ' Number between 1 and 5, inclusive.
   Debug.Print "Between 1 and 5"
' The following is the only Case clause that evaluates to True.
Case 6, 7, 8    ' Number between 6 and 8.
   Debug.Print "Between 6 and 8"
Case 9 To 10    ' Number is 9 or 10.
Debug.Print "Greater than 8"
Case Else    ' Other values.
   Debug.Print "Not between 1 and 10"
End Select
```