USEFUL VBA FUNCTIONS GROUPED BY TYPE

Arrays

The following functions operating on arrays can be used.

1. Array Function

Returns a Variant containing an array; the syntax is:

```
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.

Example

```
Dim MyWeek As Variant
Dim MyDay As String
MyWeek = Array("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")
' Return values assume lower bound set to 1
MyDay = MyWeek(2)     ' MyDay contains "Tue".
MyDay = MyWeek(4)     ' MyDay contains "Thu".
```

2. Lbound

Returns a Long containing the smallest available subscript for the indicated dimension of an array; the syntax is:

```
LBound (arrayname [, dimension])
```

where:

- arrayname is the name of the array variable,
- the optional dimension parameter is a whole number indicating which dimension's lower bound is returned. Use 1 for the first dimension, 2 for the second, and so on.

3. Ubound

Returns a Long containing the upper limit of an array dimension; the syntax is the following one:

```
UBound (arrayname [, dimension])
```

4. Filter

Returns a subset of a supplied string array, based on supplied criteria; the syntax is:

```
Filter(SourceArray, Match, [Include], [Compare])
```

Where the function arguments are:

SourceArray	-	The original array of Strings, that you want to filter.
Match	-	The string that you want to search for, within each element of the supplied SourceArray.
[Include]	-	An option boolean argument that specifies whether the returns array should consist of elements that include or do not include the supplied Match String. This can have the value True or False, meaning: *True* - Only return elements that include the Match String *False* - Only return elements that do not include the Match String If the [Include] argument is omitted, it takes on the default value True.
[Compare]	-	An optional argument, specifying the type of String comparison to make. This can be any of the following values: *vbBinaryCompare* - performs a binary comparison *vbTextCompare* - performs a text comparison *vbDatabaseCompare* - performs a database comparison

Example

```
' Filter an array of names for entries that contain "Smith".
' First create the original array of names.
Dim names As Variant
names = Array("Ann Smith", "Barry Jones", "John Smith", "Stephen
Brown", "Wilfred Cross" )
' Use the Filter function to extract names containing "Smith".
Dim smithNames As Variant
smithNames = Filter(names, "Smith" )
' The array smithNames now has length 2 and contains the Strings
' "Ann Smith" and "John Smith".
```

5. Join

Joins together an array of substrings and returns a single string; the syntax is:

```
Join(SourceArray, [Delimiter])
```

Where the function arguments are:

SourceArray	-	The array of substrings that you want to join together.
[Delimiter]	-	The delimiter that is used to separate each of the substrings when making up the new string. If omitted, the [Delimiter] is set to be a space " ".

Example

The following VBA code joins together the strings "John", "Paul" and "Smith".

```
' Join together the strings "John", "Paul" and "Smith".
Dim fullName As String
Dim names(0 to 2 ) As String
names(0) = "John"
names(1) = "Paul"
names(2) = "Smith"
fullName = Join(names)
' The variable fullName is now set to "John Paul Smith"
```

6. Split

Splits a string into a number of substrings and returns a one-dimensional array of substrings; the syntax is:

Split(Expression, [Delimiter], [Limit], [Compare])

Where the function arguments are:

Expression	-	The text string that you want to split.
[Delimiter]	-	The delimiter that is to be used to specify where the supplied Expression should be split. If omitted, the [Delimiter] is set to be a space " ".
[Limit]	-	An optional integer argument, specifying the maximum number of substrings to be returned. If the [Limit] argument is omitted, it has the default value -1, denoting that all substrings should be returned.
[Compare]	-	An optional VbCompareMethod enumeration value, specifying the type of comparison that should be used for the substrings. This can have any of the following values: *vbBinaryCompare* - performs a binary comparison *vbTextCompare* - performs a text comparison *vbDatabaseCompare* - performs a database comparison

Example

The following code split the path of a file into 4 sub-strings, assigned to the 4 elements of an array.

```
' Split the string "C:\Users\My Documents\File.txt" into substrings.
Dim substrings() As String
substrings = Split( "C:\Users\My Documents\File.txt", "\" )
' The array "substrings" now has length 4, and contains the values
' "C:", "Users", "My Documents" and "File.txt"
```

TEXT

The following functions operating on strings are available; in these notes we will only mention the Format function; for more details on the other ones please refer to the notes on "VBA programming".

VBA Text Functions	
Format	Applies a format to an expression and returns the result as a string.
InStr	Returns the position of a substring within a string.
InStrRev	Returns the position of a substring within a string, searching from right to left.
Left	Returns a substring from the start of a supplied string.
Len	Returns the length of a supplied string.
LCase	Converts a supplied string to lower case text.
LTrim	Removes leading spaces from a supplied string.
Mid	Returns a substring from the middle of a supplied string.
Replace	Replaces a substring within a supplied text string.
Right	Returns a substring from the end of a supplied string.
RTrim	Removes trailing spaces from a supplied string.
Space	Creates a string consisting of a specified number of spaces.
StrComp	Compares two strings and returns an integer representing the result of the comparison.
StrConv	Converts a string into a specified format.
String	Creates a string consisting of a number of repeated characters.
StrReverse	Reverses a supplied string.
Trim	Removes leading and trailing spaces from a supplied string.
UCase	Converts a supplied string to upper case text.

Format

Format applies a specified format to an expression and returns the result as a string; the syntax is:

```
Format(Expression,[Format],[FirstDayOfWeek],[FirstWeekOfYear])
```

Where the function arguments are:

Expression	-	The expression that you want to format.
[Format]	-	An optional argument specifying the format that is to be applied to the Expression. This can be a user-defined format or one of the predefined named formats listed below:

Predefined Date Formats:

Format		**Description**
General Date	-	Displays a date as defined in your system's General Date setting. If a date only, no time is displayed; If a time only, no date is displayed.
Long Date	-	Displays a date as defined in your system's Long Date settings.
Medium Date	-	Displays a date as defined in your system's Medium Date settings.
Short Date	-	Displays a date as defined in your system's Short Date settings.
Long Time	-	Displays a time as defined in your system's Long Time settings.
Medium Time	-	Displays a time as defined in your system's Medium Time settings.
Short Time	-	Displays a time as defined in your system's Short Time settings.

Predefined Number Formats:

Format		**Description**
General Number	-	Displays a number as it is entered.
Currency	-	Displays a number with a currency symbol, using the thousand separator and decimal places, as defined in your system's currency setting.
Euro	-	Displays a number as a currency, with the euro currency symbol.

Fixed	-	Displays at least one digit to the left of the decimal place and follows the system settings for the number of decimal places to the right of the decimal place.
Standard	-	Displays the thousand separator and follows the standard system settings for the number of digits displayed at either side of the decimal place.
Percent	-	Displays a number multiplied by 100 and followed by the percent symbol; The format follows the standard system settings for the number of digits displayed at either side of the decimal place.
Scientific	-	Displays a number using scientific notation.
Yes/No	-	Displays No if the number is equal to zero or Yes otherwise.
True/False	-	Displays False if the number is equal to zero or True otherwise.
On/Off	-	Displays Off if the number is equal to zero, or On otherwise.

[FirstDayOfWeek]	-	An optional FirstDayOfWeek enumeration value, specifying the weekday that should be used as the first day of the week. This can have any of the following values:

vbUseSystemDayOfWeek	-	The first day of the week is as specified in your system settings
vbSunday	-	Sunday
vbMonday	-	Monday
vbTuesday	-	Tuesday
vbWednesday	-	Wednesday
vbThursday	-	Thursday
vbFriday	-	Friday
vbSaturday	-	Saturday

[FirstWeek OfYear]	**-**	An optional FirstWeekOfYear enumeration value, specifying the week that should be used as the first week of the year. This can have any of the following values: *vbSystem* - The first week of the year is as specified in your system settings *vbFirstJan1* - The week in which Jan 1st occurs *vbFirstFourDays* - The first week that contains at least four days in the new year *vbFirstFullWeek* - The first full week in the new year

Example #1 – Formatting dates

The following code shows how the VBA Format function can be used to format date and time in five different ways. Specifically, in case of user defined format, the following symbols can be used, for date and time, respectively:

Symbol	Range
d	1-31 (Day of month, with no leading zero)
dd	01-31 (Day of month, with a leading zero)
w	1-7 (Day of week, starting with Sunday = 1)
ww	1-53 (Week of year, with no leading zero; Week 1 starts on Jan 1)
m	1-12 (Month of year, with no leading zero, starting with January = 1)
mm	01-12 (Month of year, with a leading zero, starting with January = 01)
mmm	Displays abbreviated month names (Hijri month names have no abbreviations)
mmmm	Displays full month names
y	1-366 (Day of year)
yy	00-99 (Last two digits of year)
yyyy	100-9999 (Three- or Four-digit year)

Symbol	Range
h	0-23 (1-12 with "AM" or "PM" appended) (Hour of day, with no leading zero)
hh	00-23 (01-12 with "AM" or "PM" appended) (Hour of day, with a leading zero)
n	0-59 (Minute of hour, with no leading zero)
nn	00-59 (Minute of hour, with a leading zero)
m	0-59 (Minute of hour, with no leading zero). Only if preceded by h or hh
mm	00-59 (Minute of hour, with a leading zero). Only if preceded by h or hh
s	0-59 (Second of minute, with no leading zero)
ss	00-59 (Second of minute, with a leading zero)

Other S.	Meaning
:	Time Separator
/	Date Separator
AM/PM	12 hours clock with upper case AM (or PM)
am/pm	12 hours clock with lower case AM (or PM)

```
' Format date/time in different ways.
Dim dt1 As String
Dim MyTime As Date, MyDate As Date
MyTime = #17:04:23#
MyDate = #January 27, 1993#
dt1 = Format( #12/31/2015 12:00:00 PM#)
' dt1 is now equal to the String "12/31/2015 12:00:00 PM".
dt1 = Format( #12/31/2015 12:00:00 PM#, "Long Date" )
' dt1 is now equal to the String "Thursday, December 31, 2015".
dt1 = Format( #12/31/2015 12:00:00 PM#, "Medium Time" )
' dt1 is now equal to the String "12:00 PM".
dt1 = Format( #12/31/2015 12:00:00 PM#, "mm/dd/yyyy" )
' dt1 is now equal to the String "12/31/2015".
dt1 = Format( #12/31/2015 12:00:00 PM#, "dddd mm/dd/yyyy hh:mm:ss" )
' dt1 is now equal to the String "Thursday 12/31/2015 12:00:00".
dt1 = Format(MyTime, "hh:mm:ss am/pm")
' dt1 is now equal to "05:04:23 pm".
dt1  = Format(MyTime, "hh:mm:ss AM/PM")
' dt1 is now equal to "05:04:23 PM".
dt1 = Format(MyDate, "dddd, mmm d yyyy")
' dt1 is now equal to "Wednesday, Jan 27 1993".
```

Note that, in the above examples:

- In the first call to the Format function, the [Format] argument is omitted. Therefore, the current system default date/time format is used.
- The second and third calls to the Format function use the predefined formats "Long Date" and "Medium Time". These format definitions will vary, according to your system date/time settings.
- The last calls to the Format function have been supplied with user-defined formats.

Example #2 - Formatting numbers

As for dates, also to format numbers some special characters can be used as "placeholder". These are:

Character	Description
None	Display the number with no formatting.
0	*Digit placeholder*. *Display a digit or a zero*. If the expression has a digit in the position where the 0 appears in the format string, display it; otherwise, display a zero in that position.
#	*Digit placeholder. Display a digit or nothing*. If the expression has a digit in the position where the # appears in the format string, display it; otherwise, display nothing in that position.
.	*Decimal placeholder*
%	*Percentage placeholder*. The expression is multiplied by 100. The percent character (%) is inserted in the position where it appears in the format string.
,	*Thousand separator*
:	*Time separator*
/	*Date separator*
E- E+ e- e+	*Scientific format*. If the format expression contains at least one digit placeholder (0 or #) to the right of E-, E+, e-, or e+, the number is displayed in scientific format and E or e is inserted between the number and its exponent.
- + $ ( )	*Display a literal character*. To display a character other than one of those listed, precede it with a backslash (\) or enclose it in double quotation marks (" ").
\	*Display the next character in the format string*. To display a character that has special meaning as a literal character, precede it with a backslash (\). The backslash itself isn't displayed. Using a backslash is the same as enclosing the next character in double quotation marks. To display a backslash, use two backslashes (\\). Examples of characters that can't be displayed as literal characters are the date-formatting and time-formatting characters (a, c, d, h, m, n, p, q, s, t, w, y, /, and :), the numeric-formatting characters (#, 0, %, E, e, comma, and period), and the string-formatting characters (@, &;, <, >, and !).
"ABC"	*Display the string inside the double quotation marks (" ")*.

Also, a user-defined format expression for numbers can have from one to four sections separated by semicolons, for instance "$#,##0;($#,##0)" is a format with two sections.
Specifically, if you use:

- *One section* – the format expression applies to all values;
- *Two sections* – the first applies to positive, the second to negative values;
- *Three sections* – as above and the third section applies to zeros;
- *Four sections* – as above, and the forth section applies to Null values

```
' Format numbers  in different ways.
Dim MyStr As String,
MyStr = Format( 50000 )' equal to the String "50000".
MyStr = Format( 50000, "Currency" )' equal to the String "$50,000.00".
MyStr = Format( 0.88, "Percent" )' equal to the String "88.00%".
MyStr = Format( 50000, "#,##0.0" )' to the String "50,000.0".
MyStr = Format( 0.88, "0.0" )' equal to the String "0.9".
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%".
```

In the following VBA code, the VBA Format function is used to format numeric values in different ways.
Note that, in the above examples:

- In the first call to the Format function, the [Format] argument is omitted. Therefore, the current system default number format is used.
- The second and third calls to the Format function use the predefined formats "Currency" and "Percentage". These format definitions may vary, according to your system settings.
- The last calls to the Format function have been supplied with user-defined formats.

Example #3 – Formatting strings

The following example shows the VBA Format function used with different format options for Strings. Note that, in this case you can use any of the following characters to create a format expression for strings.

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

```
' Format two text strings using different user-defined formats.
Dim str1 As String,str2 As String
str1 = Format( "John Smith", ">" )' str1 is now "JOHN SMITH".
str2 = Format( "123456789", "@@@-@@@-@@@" )' str2 is  "123-456-789".
```

In the above VBA code:

- In the first call to the Format function, the ">" character forces the supplied text to be formatted in upper case.
- In the second call to the Format function, the "@" character represents a displayed character and the "-" character is text to be inserted in the specified positions of the String.

Conversions

The following functions can be used to perform type casting, or other conversions operations. Most of them are self-explaining, so we will focus only on the trickiest ones.

Function	Effect
Asc	Returns an integer representing the code for a supplied character.
CBool	Converts an expression to a Boolean data type.
CByte	Converts an expression to a Byte data type.
CCur	Converts an expression to a Currency data type.
CDate	Converts an expression to a Date data type.
CDbl	Converts an expression to a Double data type.
CDec	Converts an expression to a Decimal data type.
Chr	Returns the character corresponding to a supplied character code.
CInt	Converts an expression to an Integer data type.
CLng	Converts an expression to a Long data type.
CSng	Converts an expression to a Single data type.
CStr	Converts an expression to a String data type.
CVar	Converts an expression to a Variant data type.
FormatCurrency	Applies a currency format to an expression and returns the result as a string.
FormatDateTime	Applies a date/time format to an expression and returns the result as a string.
FormatNumber	Applies a number format to an expression and returns the result as a string.
FormatPercent	Applies a percentage format to an expression and returns the result as a string.
Hex	Converts a numeric value to hexadecimal notation and returns the result as a string.
Oct	Converts a numeric value to octal notation and returns the result as a string.
Str	Converts a numeric value to a string.
Val	Converts a string to a numeric value.

1. FormatCurrency

FormatCurrency function applies a currency format to a numeric expression and returns the result as a string.

The syntax of the function is:

```
FormatCurrency(Expression, [NumDigitsAfterDecimal],
          [IncludeLeadingDigit],[UseParensForNegativeNumbers],
          [GroupDigits])
```

Where the function arguments are:

Expression	-	The numeric expression that you want to format.
[NumDigitsAfterDecimal]	-	An optional numeric value specifying the number of digits that should be displayed after the decimal. If [NumDigitsAfterDecimal] is omitted, it defaults to the value -1, denoting that the computer's regional settings should be used.
[IncludeLeadingDigit]	-	An optional vbTriState enumeration value, specifying whether a leading zero should be displayed for fractional values. This can have any of the following values: *vbFalse* - No leading zero. *vbTrue* - Display a leading zero. *vbUseDefault* - Default computer settings
[UseParensForNegativeNumbers]	-	An optional vbTriState enumeration value, specifying whether negative numbers should be encased in parentheses. This can have any of the following values: *vbFalse* - Do not encase *vbTrue* - Encase *vbUseDefault* - Use the default computer settings.

[GroupDigits]	-	An optional vbTriState enumeration value, specifying whether the number should be grouped (into thousands, etc.), using the group delimiter that is specified on the computer's regional settings. This can have any of the following values: *vbFalse* - Do not group digits. *vbTrue* - Group digits. *vbUseDefault* - Use the default computer settings.

Example

The following example shows how the VBA FormatCurrency function can be used to format numeric values as currencies. Each example uses different formatting rules.

```
' Format different numeric values as currencies.
Dim cur1 As String, cur2 As String, cur3 As String, cur4 As String
cur1 = FormatCurrency( 1000000.00 )
' cur1 is now equal to the String "$1,000,000.00".
cur2 = FormatCurrency( 1000000.00, , , , vbFalse )
' cur2 is now equal to the String "$1000000.00".
cur3 = FormatCurrency( 100.55, 0 )
' cur3 is now equal to the String "$101".
cur4 = FormatCurrency( -500, 2, , vbTrue )
' cur4 is now equal to the String "($500.00)".
```

Note that in the above examples:

- In the first call to the FormatCurrency function, only the Expression argument has been provided. Therefore, the returned string "$1,000,000.00" has used the default currency format for the current computer.
- In the second call to the FormatCurrency function, the [GroupDigits] argument is set to vbFalse. Therefore, the number 1,000,000.00 is returned as the string "$1000000.00" (with no commas separating the groups of numbers).

- In the third call to the FormatCurrency function, the [NumDigitsAfterDecimal] argument is set to 0. Therefore, the value 100.55 is rounded to zero decimal places before being returned as the currency string "$101".
- In the fourth call to the FormatCurrency function, the [UseParensForNegativeNumbers] argument is set to vbTrue. Therefore, the value -500 is returned as the string ($500.00).

2 FormatDateTime

FormatDateTime function applies a date and/or time format to an expression and returns the result as a string. The syntax of the function is:

```
FormatDateTime(Expression, [NamedFormat])
```

Where the function arguments are:

Expression	-	The expression that you want to format.
[NamedFormat]	-	An optional vbDateTimeFormat enumeration specifying the format that is to be applied to the Expression. This can be any of the following:

Format		Description
vbGeneral	-	Displays a date and/or time as defined in your system's General Date setting. If a date only, no time is displayed; If a time only, no date is displayed.
vbLongDate	-	Displays a date as defined in your system's Long Date settings.
vbLongTime	-	Displays a time as defined in your system's Long Time settings.
vbShortDate	-	Displays a date as defined in your system's Short Date settings.
vbShortTime	-	Displays a time as defined in your system's Short Time settings.

Example

```
The following example shows how to format both data and time.'
Format the date and time 1/1/2016 12:00:00 in different ways.
Dim dt1 As String. dt2 As String.dt3 As String
dt1 = FormatDateTime( #1/1/2016 12:00:00 PM# )
' dt1 is now equal to the String "1/1/2016 12:00:00 PM".
dt2 = FormatDateTime( #1/1/2016 12:00:00 PM#, vbLongTime )
' dt2 is now equal to the String "12:00:00 PM".
dt3 = FormatDateTime( #1/1/2016 12:00:00 PM#, vbShortDate )
' dt3 is now equal to the String "1/1/2016".
```

3 FormatNumber

FormatNumber function applies a number format to a numeric expression and returns the result as a string.The syntax of the function is:

FormatNumber(Expression,[NumDigitsAfterDecimal], [IncludeLeadingDigit],[UseParensForNegativeNumbers], [GroupDigits])

Where the function arguments are the same as the FormatCurrency function.

```
' Format numeric values in different ways.
Dim num1 As String
Dim num2 As String
Dim num3 As String
Dim num4 As String
num1 = FormatNumber( 1000000 )
' num1 is now equal to the String "1,000,000.00".
num2 = FormatNumber( 1000000,   ,   ,   , vbFalse )
' num2 is now equal to the String "1000000.00".
```

4 FormatPercent

FormatPercent function applies a percentage format to a numeric expressioand returns the result as a string.

The syntax of the function is:

```
FormatPercent(Expression,[NumDigitsAfterDecimal],
        [IncludeLeadingDigit],[UseParensForNegativeNumbers],
                                              [GroupDigits])
```

Where the function arguments are the same as before.

```
' Format numeric values in different percentage formats.
Dim pc1 As String
Dim pc2 As String
Dim pc3 As String
Dim pc4 As String
pc1 = FormatPercent( 10 )
' pc1 is now equal to the String "1,000.00%".
pc2 = FormatPercent( 10,   ,   ,   , vbFalse )
' pc2 is now equal to the String "1000.00%".
pc3 = FormatPercent( 0.559, 0 )
' pc3 is now equal to the String "56%".
pc4 = FormatPercent( -0.5,   ,   , vbTrue )
' pc4 is now equal to the String "(50.00%)".
```

5 Val

The Val function converts a supplied string into a numeric value.

The syntax of the function is:

```
Val(String)
```

Where the String argument is the string that you want to convert into a number.

Note:

- The Val function ignores spaces in the supplied String, but continues to read the characters after any space(s).
- If the Val function encounters a character that is not recognised as part of a number, it stops reading the String at this point. Characters that cannot be recognised as parts of numbers include currency symbols, the % symbol and commas.

Example 1 – Standard functioning

```
num1 = Val( "500" )
' num1 is now equal to 500.
num2 = Val( "+10.9" )
' num2 is now equal to 10.9.
num3 = Val( "-0.5" )
' num3 is now equal to -0.5.
num4 = Val( "10.5 km" )' num4 is now equal to the String "10.5".
```

Example 2 – Unexpected behaviours

As the Val function ignores spaces and also ignores everything after the first character that is not recognised as part of a number, you may sometimes get results that you do not expect. Some examples are shown below.

```
num1 = Val( "10     10" )
' num1 is now equal to 1010 (spaces are ignored).
num2 = Val( "$500.00" )
' num2 is now equal to 0
' all characters after the $ symbol are ignored).
num3 = Val( "1,000" )' num3 is now equal to 1
' all characters after the comma are ignored.
```

Note that in the above examples:

- If the supplied String contains spaces, the Val function ignores these, and continues to read the remaining characters.
- If the supplied String starts with a $ symbol, this is not recognised as part of a number, and so the Val function ignores this, and all the remaining characters, and returns the value 0.
- If the supplied String contains one or more commas, these are not recognised as part of a number, and so the Val function ignores the first comma and all characters after this.

Math Functions

The following math functions are available

VBA Math & Trigonometric Functions	
Abs	Returns the absolute value of a number.
Atn	Calculates the arctangent of a supplied number.
Cos	Calculates the cosine of a supplied angle.
Exp	Calculates the value of e^x for a supplied value of x.
Fix	Truncates a number to an integer (rounding negative numbers towards zero).
Int	Returns the integer portion of a number (rounding negative numbers).
Log	Calculates the natural logarithm of a supplied number.
Rnd	Generates a random number between 0 and 1.
Round	Rounds a number to a specified number of decimal places.
Sgn	Returns an integer representing the arithmetic sign of a number.
Sin	Calculates the sine of a supplied angle.
Tan	Calculates the tangent of a supplied angle.
Sqr	Returns the square root of a number.

Information Functions

The following math functions are available in VBA:

VBA Information Functions	
IsArray	Tests if a supplied variable is an array.
IsDate	Tests if a supplied expression is a date.
IsEmpty	Tests if a supplied variant is Empty.
IsError	Tests if a supplied expression represents an error.
IsMissing	Tests if an optional argument to a procedure is missing.
IsNull	Tests if a supplied expression is Null.
IsNumeric	Tests if a supplied expression is numeric.
IsObject	Tests if a supplied variable represents an object variable.
TypeName	Provides information about a variable
VarType	Returns an Integer indicating the subtype of a variable.

1 TypeName Function

This VBA function receives in input a variable and it returns, as output, a string containing information about the input variable. Its syntax is the following one:

```
TypeName (varname)
```

The required varname argument is a variable of any type, except a variable of a user defined type.

Example

The following example uses the TypeName function to get information about a variable.

```
Dim NullVar As Variant, MyType As Variant
Dim StrVar As String
Dim IntVar As Integer
Dim CurVar As Currency
Dim ArrayVar (1 To 5) As Integer
NullVar = Null     ' Assign Null value.
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()".
```

2 VarType Function

This VBA function receives in input a variable and it returns, as output, an integer indicating the subtype of a variable. containing information about the input variable. Returned values are coded as Enum Type and can be easily found on the web and/or in the VBA Development Environment. For instance:

- vbEmpty (corresponds to 0) and indicate an empty variable;
- vbNull (corresponds to 1) and indicate a null variable;
- vbInteger(corresponds to 2) and indicate an integer;
- …
- vbDate (corresponds to 7) and indicate a date/time;
- vbString (corresponds to 8) and indicate a string.
- …

Its syntax is the following one:

```
VarType (varname)
```

The required varname argument is a variable of any type, except a variable of a user defined type.

Example

The following example uses the VarType function to determine the subtype of a variable.

```
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.
```

ERROR FUNCTIONS

The following functions operating on error types can be used.

1 CVERR

The CVErr function returns an Error data type, relating to a user-specified error code. The syntax is:

```
CVErr(Expression)
```

Where the supplied Expression is the required error code.

Example

The following example shows a simple VBA function that divides a supplied number by a second supplied number. If the second supplied number is zero, the CVErr function is used to create an Error data type.

```
' Function to divide two numbers.
Function performDiv(num1 As Double, num2 As Double)
  If num2 = 0 Then
   ' Return Error data type for error code 11
   ' i.e., division by zero
     performDiv = CVErr(11) ' performDiv is now equal to Error 11.
  Else
   ' Perform the division.
     performDiv = num1 / num2
  End If
End Function
```

In the above function, if the second supplied number is zero, the function returns the error data type 'Error 11' (which represents a division by zero).

2 ERROR

Error returns the error message corresponding to a supplied error code.The syntax of the function is:

```
Error([ErrorNumber])
```

Where [ErrorNumber] is an optional integer argument representing the required error number.

Note that;

- If the [ErrorNumber] argument is omitted, the Error function returns the most recent run-time error;

- If the [ErrorNumber] argument is zero, the Error function returns an empty String;
- If the [ErrorNumber] argument is omitted and there have not been any run-time errors so far, the Error function returns an empty String.

Example 1

The following VBA code uses the Error function to get the error messages for the error codes 5 and 11.
The last call to the Error function has no argument and so returns the most recent run-time error (none in this case).

```
' Display the error messages for different error codes.
Dim errMsg1 As StringDim,errMsg2 As StringDim, errMsg3 As String
errMsg1 = Error(5)' errMsg1 = "Invalid procedure call or argument"
errMsg2 = Error(11)'errMsg2 = "Division by zero"
errMsg3 = Error()' errMsg3 = "" (no run-time errors have yet occurred)
```

Example 2

The following example shows a simple VBA function that divides a supplied number by a second supplied number. If the second supplied number is zero, a message box showing the corresponding error message is displayed.

```
' Function to divide two numbers.
Function performDiv(num1 As Double,num2 As Double )
    If num2 = 0 Then
   ' Display the error message corresponding to error code 11
        MsgBox (Error( 11 ))' Code to run if the divisor is zero
        performDiv = CVErr(11)
    Else
   ' Perform the division
        performDiv = num1 / num2
    End If
End Function
```

PROGRAM FLOW

Instead of using nested If … Then … Else and/or Selec … Case statements, the following program folw functions can be used.

1 CHOOSE

For a supplied index, the Choose function selects the corresponding value from a list of arguments. The syntax is:

```
Choose(Index, [Choice-1], [Choice-2], ... )
```

Where the function arguments are:

Index	-	The index of the value that you want to return (it must be between 1 and n, where n is the number of possible values). If the supplied Index is less than 1, or is greater than the number of supplied choices, the Choose function returns the value Null.
[Choice-1], **[Choice-2],** **…**	-	A list of possible values to be returned (depending on the value of Index)

Example

```
' Return specified values from a list of names
Dim val1, val2, val3, val4
val1 = Choose(1, "Mary", "Joseph", "Lucy", "Peter" )
' val1 is now equal to "Mary"
val2 = Choose(2, "Mary", "Joseph", "Lucy", "Peter" )
' val2 is now equal to "Joseph"
val3 = Choose(3, "Mary", "Joseph", "Lucy", "Peter" )
' val13 is now equal to "Lucy"
val4 = Choose(4, "Mary", "Joseph", "Lucy", "Peter" )
' val4 is now equal to "Peter"
```

The above VBA code uses the Choose function to return a value from the list "Mary", "Joseph", "Lucy", "Peter".

The value of the Index argument determines which of the list items is returned:

- When Index is set to 1, the Index function returns the first list item, "Mary";
- When Index is set to 2, the Index function returns the second list item, "Joseph";

- When Index is set to 3, the Index function returns the third list item, "Lucy";
- When Index is set to 4, the Index function returns the fourth list item, "Peter".

2 IFF

The VBA IIf function evaluates an expression and returns one of two values, depending on whether the expression evaluates to True or False. The syntax is:

```
IIf( Expression, TruePart, FalsePart )
```

Where the function arguments are:

Expression	-	The expression that is to be evaluated.
TruePart	-	The value that is to be returned if the supplied Expression evaluates to True.
FalsePart	-	The value that is to be returned if the supplied Expression evaluates to False.

Example 1

```
' Test if a Supplied Integer is Positive or Negative
Dim testVal As IntegerDim. sign1 As StringDim, sign2 As String
' First call to IIf function. The test value is negative
testVal = -2
sign1 = IIf(testVal < 0, "negative", "positive" )
' sign1 is now equal to "negative"
' Second call to IIf function. The test value is positive
testVal = 8
sign2 = IIf(testVal < 0, "negative", "positive" )
' sign2 is now equal to "positive"
```

In the above VBA code:

- In the first call to the IIf function, the expression, testVal < 0 evaluates to True and so the TruePart argument (the String "negative") is returned;
- In the second call to the IIf function, the expression, testVal < 0 evaluates to False and so the FalsePart argument (the String "positive") is returned.

Example 2

The following VBA code shows a nested IIf function.

```
' Test if a Supplied Integer is Positive, Negative or Zero
Dim testVal As Integer
Dim sign1 As String
testVal = -2
sign1=IIf(testVal=0, "zero", IIf(testVal < 0,"negative","positive" ))
' sign1 is now equal to "negative".
```

In the above VBA code, there are two calls to the IIf function, one of which is nested inside the other:

- If the outer IIf function evaluates to True, the String "zero" is returned;
- If the outer IIf function evaluates to False, the inner IIf function is called:
    - If the inner IIf function evaluates to True, the String "negative" is returned;
    - If the inner IIf function evaluates to False, the String "positive" is returned.

In the example, the test value is equal to -2 and so the nested IIf function returns the String "negative".

3 SWITCH

The Switch function evaluates a list of Boolean expressions and returns a value associated with the first true expression.The syntax is:

```
Switch(Expr_1, Val_1, [Expr_2, Val_2], [Expr_3, Val_3], ... )
```

Where the function arguments are:

Expr_1, [Expr_2], ...	-	One or more boolean expressions to be evaluated
Val_1, [Val_2], ...	-	The values to be returned if the corresponding Expr-1, [Expr-2], etc. is the first True expression. If none of the supplied expressions evaluate to True, the Switch function returns the value Null.

Example 1

```
' Return a surname corresponding to a supplied forename
Dim fname As String
Dim sname As String
fname = "Lucy"
sname = Switch(fname = "Mary","Jones",fname="Joseph","Johnson",_
                                       _ fname = "Lucy", "Smith" )
' sname is now equal to "Smith"
```

In the above call to the Switch function, the third expression, fname ="Lucy" is the first expression to evaluate to True. Therefore, the function returns the associated surname, "Smith".

Example 2

```
' Return a name, depending on an integer value.
Dim i As Integer
Dim fname As String
i = 12
fname = Switch(i< 10,"Mary",i< 20,"Joseph",i<30,"Lucy")
'fname is now equal to "Joseph"
```

In the above call to the Switch function, the second expression, i<20 is the first expression to evaluate to True. Therefore, the function returns the associated name, "Joseph".

Note that both the second expression i<20 and the third expression, i<30 evaluate to True. However, the Switch function only returns the value corresponding to the first True expression in the supplied list.

DATES

As well as functions operating on strings, also functions operating on date and time are extremely useful to manipulate data stored in a Relational Data Base. The main ones are listed in the table below; in the following parts we will detail the functioning of the less intuitive ones.

VBA Date & Time Functions	
Date	Returns the current date.
DateAdd	Adds a time interval to a date and/or time.
DateDiff	Returns the number of intervals between two dates and/or times.
DatePart	Returns a part (day, month, year, etc.) of a supplied date/time.
DateSerial	Returns a Date from a supplied year, month and day number.
DateValue	Returns a Date from a String representation of a date/time.
Day	Returns the day number (from 1 to 31) of a supplied date.
Hour	Returns the hour component of a supplied time.
Minute	Returns the minute component of a supplied time.
Month	Returns the month number (from 1 to 12) of a supplied date.
MonthName	Returns the month name for a supplied month number (1 to 12).
Now	Returns the current date and time.
Second	Returns the second component of a supplied time.
Time	Returns the current time.
Timer	Returns the number of seconds that have elapsed since midnight.
TimeSerial	Returns a Time from a supplied hour, minute and second.
TimeValue	Returns a Time from a String representation of a date/time.
Weekday	Returns an integer (from 1 to 7), representing the weekday of a supplied date.
WeekdayName	Returns the weekday name for a supplied integer (from 1 to 7).
Year	Returns the year of a supplied date.

1 DateAdd

DateAdd adds a time interval to a supplied date and/or time, and returns the resulting date/time. The syntax is:

```
Dateadd(Interval, Number, Date )
```

Where the function arguments are:

Interval	-	A string specifying the interval to be used. This can have any of the following values: "d" - Days "h" - Hours "n" - Minutes "m" - Months "q" - Quarters (of a Year) "s" - Seconds "ww" - Weeks "yyyy" - Years
Number	-	The number of intervals to add to the specified Date.
Date	-	The original date/time that you want to add the specified number of intervals to.

Example

```
Dim oldDate As Date
Dim newDate As Date

' Add 32 days to the date 11/29/2015
oldDate = #11/29/2015#
newDate = DateAdd("d", 32, oldDate )' the date #12/31/2015#

' 27 hours after 9:00 AM on 11/29/2015
oldDate = #11/29/2015 9:00:00 AM#
newDate = DateAdd("h", 27, oldDate ) ' the date #11/30/2015 12:00#

' Calculate date that is 3 months after 12/31/2015
oldDate = #12/31/2015#
newDate = DateAdd( "m", 3, oldDate )' the date #3/31/2016#
```

2 DateDiff

DateDiff returns a Long data value representing the number of intervals between two supplied dates/times. The type of interval (e.g. hours, days, months, etc.) is specified by the user. The syntax is:

```
DateDiff(Interval,Date1,Date2,[FirstDayOfWeek],_
                                        _ FirstWeekOfYear])
```

Where the function arguments are:

Interval	-	A string specifying the interval to be used. This can have any of the following values (same values as in the previous table)
Date1	-	A date value, representing the start date/time for the calculation.
Date2	-	A date value, representing the end date/time for the calculation.
[FirstDayOfWeek]	-	An optional FirstDayOfWeek enumeration value, specifying the weekday that should be used as the first day of the week. This can have any of the following values:

vbUseSystemDayOfWeek	-	as specified in your system settings
vbSunday	-	Sunday
vbMonday	-	Monday
vbTuesday	-	Tuesday
vbWednesday	-	Wednesday
vbThursday	-	Thursday
vbFriday	-	Friday
vbSaturday	-	Saturday

[FirstWeekOfYear]	-	An optional FirstWeekOfYear enumeration value, specifying the week that should be used as the first week of the year. This can have any of the following values: *vbSystem* - The first week as specified in system settings *vbFirstJan1* - The week in which Jan 1st occurs *vbFirstFourDays* - The first week that contains at least four days in the new year *vbFirstFullWeek* - The first full week in the new year

Example

```
' Calculate the number of days between 11/29/2015 and 12/31/2015
Dim dt1 As Date, dt2 As Date
Dim nDays As Long, nHours As Long, nWeeks As Long
dt1 = #11/29/2015#
dt2 = #12/31/2015#
nDays = DateDiff("d", dt1, dt2)' nDays now has the value 32

' Number of hours between 11/29/2015 9:00 and 11/30/2015 12:00
dt1 = #11/29/2015 9:00:00 AM#
dt2 = #11/30/2015 12:00:00 PM#
nHours = DateDiff("h", dt1, dt2)' nHours now has the value 27

' Number of weeks between 11/29/2015 and 12/8/2015
dt1 = #11/29/2015#
dt2 = #12/8/2015#
nWeeks = DateDiff("w", dt1, dt2)'nWeeks now has the value 1
' Calculate the number of calendar weeks between 11/29/2015 and
12/8/2015
' First day of the week = Monday
dt1 = #11/29/2015#
dt2 = #12/8/2015#
nWeeks = DateDiff("ww", dt1, dt2, vbMonday )' nWeeks is 2
```

Please note that, if the Interval argument is specified to be "w" (weeks), the DateDiff function returns the number of whole weeks between the two supplied dates. Partial weeks are ignored. Conversely, if "ww" (i.e, calendar weeks) is specified, returns the number of weeks between the start of the week containing the supplied Date1 and the start of the week containing the supplied Date2.

Example 2

```
' Calculate the number of months between 11/30/2015 and 12/1/2015
' and the number of months between 12/1/2015 and 12/31/2015
Dim nMths1 As Long, nMths2 As Long
nMths1 = DateDiff("m", #11/30/2015#, #12/1/2015# )
nMths2 = DateDiff("m", #12/1/2015#, #12/31/2015# )
' nMths1 has the value 1 and nMths2 has the value 0
```

Please note that, if the interval is specified to be months, quarters or years, the DateDiff function simply subtracts the months, quarters and/or years during which each of the supplied dates occur. Therefore, the function will return 1 for the number of months between the last day of November 2015 and the first day of December 2015, but will return 0 for the number of months between the first and last day of December 2015.

3 DatePart

DatePart returns a part (day, month, week, etc.) of a supplied date and/or time.The syntax is:

```
Month(Interval, Date, [FirstDayOfWeek], [FirstWeekOfYear])
```

Where the function arguments have the same meaning of the previously introduced functions.

Example

```
' Return the day, month & year from the date 12/31/2015
Dim dy As Integer, mth As Integer, yr As Integer
dy = DatePart("d", #12/31/2015#)
mth = DatePart("m", #12/31/2015#)
yr = DatePart("yyyy", #12/31/2015#)
' dy = 31, mth = 12 and yr = 2015.

' Return hour, minute and seconds from the time 3:05:30 PM
Dim hr As Integer, min As Integer, sec As Integer
hr = DatePart("h", #3:05:30 PM# )
min = DatePart("n", #3:05:30 PM# )
sec = DatePart("s", #3:05:30 PM# )
' hr = 15, min = 5 and sec = 30
' Return day, week & quarter info. from the date 12/31/2015
Dim dyYr As Integer, dyWk As Integer, wkYr As Integer,
Dim qtr As Integer
dyYr = DatePart("y", #12/31/2015# )
dyWk = DatePart("w", #12/31/2015# )
wkYr = DatePart("ww", #12/31/2015# )
qtr = DatePart("q", #12/31/2015# )
' dyYr = 365, dyWk = 5, wkYr = 53 and qtr = 4.
```

4 DateSerial

DateSerial returns a Date from a supplied year, month and day number. The syntax is:

```
DateSerial(Year, Month, Day)
```

Where the function arguments are:

Year	-	An integer representing the year. Note that: One- and Two-digit year numbers (from 0 to 99) are interpreted as years between 1930 and 2029; Negative year numbers are subtracted from the year 2000 (e.g. -1 represents the year 1999, etc.).
Month	-	An integer representing the month. Integer values less than 1 or greater than 12 are interpreted as follows: -1 - represents November of the previous year 0 - represents December of the previous year 13 - represents January of the following year 14 - represents February of the following year etc.
Day	-	An integer representing the day of the month. Integer values less than 1 or greater than the number of days in the current month are interpreted as follows: -1 - represents the second to the last day of the previous month 0 - represents the last day of the previous month days in current month + i - represents the i-th day of the following month

Example

```
Dim dt As DAte
dt = DateSerial (2015, 2, 3) 'Result: "2015-02-03"
dt = DateSerial (2016, 1, 0) 'Result: "2015-12-31"
dt = DateSerial (2016, 0, 0) 'Result: "2015-11-30"
dt = DateSerial (2016, -1, 1) 'Result: "2015-11-01
```

5 DateValue

DateValue returns a VBA Date from a supplied String representation of a date. Time information within the supplied string is ignored. The syntax is:

```
DateValue(Date)
```

Where the Date argument is a valid String representation of date/time.

The DateValue function can interpret text representations of dates that are in a recognised Excel format. However, the function is unable to interpret dates that include the text description of the weekday (Sunday, Monday, etc).

Example

```
' Convert two supplied strings into dates
Dim dt1 As Date, dt2 As Date
dt1 = DateValue("12/31/2015" )
' dt1 is now equal to the date 12/31/2015
dt2 = DateValue("Jan 1 2016 3:00 AM" )
' dt2 is now equal to the date 1/1/2016
```

In the above VBA code:

- The DateValue function converts the text string "12/31/2015" into the VBA date 12/31/2015
- The DateValue function converts the text string "Jan 1 2016 3:00 AM" into the VBA date 1/1/2016
- Note that the time information in this text string is ignored

6 TimeSerial

TimeSerial returns a Time from a supplied hour, minute and second. The syntax of the function is:

```
TimeSerial(Hour, Minute, Second)
```

Where the function arguments are:

Hour	-	An integer (generally from 0 to 23) representing the hour of the required time.
Minute	-	An integer (generally from 0 to 59) representing the minute of the required time. If the supplied Minute argument is less than 0 or greater than 59, this value is subtracted from, or added to, the supplied number of hours (as a partial hour).
Second	-	An integer (generally from 0 to 59) representing the second of the required time. If the supplied Second argument is less than 0 or greater than 59, this value is subtracted from, or added to, the supplied number of minutes (as a partial minute).

Example

```
' Return two simple times
Dim time1 As Date, time2 As Date, time3 As Date
time1 = TimeSerial( 11, 0, 0 )
' The variable time1 is now equal to 11:00:00 AM.
time2 = TimeSerial( 19, 15, 30 )
' The variable time2 is now equal to 7:15:30 PM.

' Return four different times (minute or second is outside the range 0-59)
time1 = TimeSerial( 8, -1, 0 )
' The variable time1 is now equal to 7:59:00 AM.
time2 = TimeSerial( 8, 60, 0 )
' The variable time2 is now equal to 9:00:00 AM.
Dim time3 As Date
time3 = TimeSerial( 8, 0, -1 )
' The variable time3 is now equal to 7:59:59 AM.
Dim time4 As Date
time4 = TimeSerial( 8, 0, 60 )
' The variable time4 is now equal to 8:01:00 AM.
```

MESSAGES

In VBA it is possible to ask the user for some inputs and, similarly, it is possible to show a message to the user and to wait for his or her answer (depending on the displayed message). To this aim it is possible to use the InputBox and the MsgBox functions. Actually, especially when programming using Forms and graphical components, the use of these functions is fairly limited, yet it is worth knowing how they work.

1 InputBox

InputBox function displays a dialog box, prompting the user for input, and containing an OK button and a Cancel button. The function returns a text string containing the user's input if the OK button is selected or an empty text string if the Cancel button is selected. The syntax is:

```
InputBox(Prompt, [Title], [Default], [XPos], [YPos], _
                                   _[HelpFile], [Context])
```

Where the function arguments are:

Prompt	-	The text string that you want to appear in the input box.
[Title]	-	A optional text string that specifies a title to be displayed at the top of the input box.
[Default]	-	A optional text string that is displayed in the input box as the default response if no other response is entered.
[XPos]	-	An integer specifying (in twips) the horizontal distance of the input box, from the left edge of the screen.
[YPos]	-	An integer specifying (in twips) the vertical distance of the input box, from the top of the screen.
[HelpFile]	-	An optional string argument, identifying the Help file relating to the input box.
[Context]	-	An optional numeric value that is the context ID for the Help topic relating to the input box.

2 MsgBox

MsgBox function displays a modal message box. The function returns a VbMsgBoxResult enumeration, which tells you which button has been selected by the user. The syntax is:

```
MsgBox(Prompt, [Buttons], [Title], [HelpFile], [Context])
```

Where the function arguments are:

Prompt	-	The text string that you want to appear in the message box.
[Buttons]	-	An optional argument that specifies the properties of the message box. The main options, defining the number and type of buttons to be displayed, are:
[Title]	-	A optional text string that specifies a title to be displayed at the top of the message box.

Value		**Buttons Displayed**
vbOKOnly	-	OK
vbOKCancel	-	OK and Cancel
vbAbortRetryIgnore	-	OK and Cancel
vbYesNoCancel	-	Yes, No and Cancel
vbYesNo	-	Yes and No
vbRetryCancel	-	Retry and Cancel

MsgBox returns one of the following VbMsgBoxResult enumeration values, informing the developer of the option that has been selected by the user:

VbMsgBoxResult	**Value**	**Button Selected**
vbOK	1	OK
vbCancel	2	Cancel
vbAbort	3	Abort
vbRetry	4	Retry
vbIgnore	5	Ignore
vbYes	6	Yes
vbNo	7	No