ACOS

Returns the arccosine of a number. The arccosine of a number is the same as the inverse cosine.

Usage

Angle = ACOS (number)

Description

The argument taken by ACOS is a cosine of an angle, and must be -1< n < 1. The return value is the angle whose cosine is the number you supply. The angle, by default, should be expressed in radians. To express in gradians or degrees, use the SetAngleMeasure function.

Example

ACOS(0.1) = 1.470629
ACOS(COS(1.47)) = 1.47

Example

See Also

ASIN , ATAN

ABS

Returns the absolute value of a number (makes it always positive).

Usage

    number = ABS(number)

Description

    Returns the absolute value of a number (makes it always positive).

Example

```
ABS(1) = 1
ABS(-1) = 1
```

Example

ACOSH

Returns the arc-hyperbolic cosine of a number.

Usage

double = ACOSH(number)

Description

Returns the arc-hyperbolic cosine of a number .The arc-hyperbolic cosine is the same as the inverse hyperbolic cosine.

The number taken by ACOSH is a hyperbolic cosine of another number, and must be a real number greater than or equal to 1. The return value is the number whose hyperbolic cosine is the number you supply.

ACOSH is defined by:
ACOSH(x) = Log(x + Sqrt(x2 - 1))

Example

ACOSH(1) = 0
ACOSH(COSH(1)) = 1

Example

See Also

ASINH , ATANH

ASIN

Returns the arcsine of a number. The arcsine of a number is the same as the inverse sine.

Usage

Angle = ASIN (number)

Description

The argument taken by ASIN is a sine of an angle, and must be -1< n < 1. The return value is the angle whose sine is the number you supply. The angle, by default, should be expressed in radians. To express in gradians or degrees, use the SetAngleMeasure function.

Example

ASIN(0.1) = 0.100167
ASIN(SIN(0.1)) = 0.1

Example

See Also

ACOS , ATAN

ASINH

Returns the arc-hyperbolic sine of a number.

Usage

double = ASINH(number)

Description

Returns the arc-hyperbolic sine of a number .The arc-hyperbolic sine is the same as the inverse hyperbolic sine.

The number taken by ASINH is a hyperbolic sine of another number, and must be a real number greater than or equal to 1. The return value is the number whose hyperbolic sine is the number you supply.

ASINH is defined by:
ASINH(x) = Log(x + Sqrt(x2 + 1))

Example

ASINH(1) = 0.881374
ASINH(ASINH(1)) = 1

Example

See Also

ACOSH , ATANH

ATAN

Returns the arctangent of a number. The arctangent of a number is the same as the inverse tangent.

Usage
  Angle = ATAN (number)

Description
  The argument taken by ATAN is a tangent of an angle. The return value is the angle whose tangent is the number you supply. The angle, by default, should be expressed in radians. To express in gradians or degrees, use the SetAngleMeasure function.

Example
  ATAN(1) = 0.785398
  ATAN(TAN(1)) = 1

Example

See Also

ASIN , ACOS

ATAN2

Returns the arctangent of a number based on X and Y coordinates. The arctangent of a number is the same as the inverse tangent.

Usage

Angle = ATAN2 (X, Y)

Description

The argument taken by ATAN2 are X and Y coordinates to a point. The behavior of ATAN2 is just like ATAN(Y/X). The return value is the angle whose tangent is the ratio to the numbers you supply. The angle, by default, should be expressed in radians. To express in gradians or degrees, use the SetAngleMeasure function.

Example

ATAN2(1,1) = 0.785398

Example

ATANH

Returns the arc-hyperbolic tangent of a number.

Usage

double = ATANH(number)

Description

Returns the arc-hyperbolic tangent of a number .The arc-hyperbolic tangent is the same as the inverse hyperbolic tangent.

The number taken by ATANH is a hyperbolic tangent of another number, and must be a real number -1< n < 1. The return value is the number whose hyperbolic tangent is the number you supply.

ATANH is defined by:
ATANH(x) = Log((1 + x) / (1 - x)) / 2

Example

ATANH(.5) = 0.549306
ATANH(ATANH(1)) = 1

Example

See Also

ASINH , ACOSH

CEILING

Rounds a number to the next higher absolute value.

Usage

number = CEILING(value, significance)

Description

Takes:

Value: any number.

Significance: Which digit to round to. Positive numbers mean the number of places to the right of the decimal. Zero means an integer. Negative values mean significant digits to the left of the decimal point.

Returns:

Number rounded up to the next higher absolute value.

Example

```
CEILING(1.3, 1) = 2
CEILING(-4.3,-1) = -5
CEILING(17.45,0) = 18
CEILING(165.165,-2) = 170
```

Example

See Also

FLOOR

COMBIN

Returns the number of combinations of groups you can form.

Usage

integer = COMBIN(number of items total, number of items in a group)

Description

A combination differs from a permutation in that the order of items does not matter.

For example, if there were four colors, and you wanted to know how man6y combinations there were, you would have the following choices:

red	blue		
red		yellow	
red			green
	blue	yellow	
	blue		green
		yellow	green

Note that colors are not matched with themselves; each item can be used only once. Also notice how red-blue and blue-red are not both considered; the change in order does not constitute a new combination.

The formula is:

Combination = (Total_Items)! / ((Items_In_Group)! * (Total_Items - Items_In_Group)!)

Example

COMBIN(4,2) = 6
COMBIN(8,3) = 56

Example

COS

Returns the cosine of an angle.

Usage

number = COS(angle)

Description

Returns a cosine. The angle, by default, should be expressed in radians. To express in gradians or degrees, use the SetAngleMeasure function.

A cosine is the ratio of the hypotenuse and the adjacent side of a right triangle. The ratio is always -1 < n < 1.

Example

COS(0) -= 1
COS(0.5) = 0.877583

Example

See Also

SIN , TAN

COSH

Returns the hyperbolic cosine of a number.

Usage

double = COSH(number)

Description

Returns the hyperbolic cosine of a number .

COSH is defined by:
COSH(x) = (ex + e-x) / 2

Example

COSH(7) = 548.317

Example

See Also

SINH , TANH

DEGREES

Converts radians to degrees.

Usage

number = DEGREES(Radians)

Description

Translates an angle measurement from radians to degrees.

The formula is:
Degrees = Radians * 180 / pi.

Example

DEGREES(3.14159) = 180
DEGREES(pi/2) = 90
DEGREES(pi/4) = 45

Example

See Also

RADIANS

EVEN

Rounds to the next higher even integer, using absolute value.

Usage

integer = EVEN(number)

Description

EVEN rounds the number to the next even number of larger absolute magnitude.

Example

EVEN(-1) = -2
EVEN(1.2) = 2
EVEN(4) = 4

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used.
The variable types are:
integer = EVEN(*double*)

Example

See Also

ODD

EXP

Returns the mathematical constant “e” raised to an exponent.

Usage

double = EXP(number)

Description

Returns the mathematical constant “e” raised to an exponent. The value “e” ios approximately 2.718282. EXP si the inverse of the natural log function (LN).

Example

EXP(1) = 2.718282
EXP(1.3) = 3.669297
EXP(LN(4)) = 4

Example

FACT

Returns a factorial.

Usage

integer = FACT(integer)

Description

A factorial is a type of series defined by:

n * (n-1) * (n-2)… 1

n must be a positive integer. By convention, the factorial of zero is one.

Example

FACT(1) = 1

FACT(0) = 1

FACT(4) = 24

Example

FIX

Rounds a number downwards using absolute value top the nearest integer.

Usage

integer = FIX(number)

Description

Rounds a number downwards using absolute value top the nearest integer.

Example

FIX(1.1) = 1

FIX(-1.1) = -1

Example

FLOOR

Rounds a number down to a specified significance.

Usage

number = FLOOR(value, significance)

Description

Takes:

Value: any number.

Significance: Which digit to round to. Positive numbers mean the number of places to the right of the decimal. Zero means an integer. Negative values mean significant digits to the left of the decimal point.

Returns:

Number rounded down to the next higher absolute value.

Example

FLOOR(1.3, 1) = 1
FLOOR(-4.3,-1) = -4
FLOOR(17.45,0) = 17
FLOOR(165.165,-2) = 160

Example

See Also

CEILING

INT

Rounds a number downwards using real value.

Usage

integer = INTEGER(number)

Description

Rounds a number downwards using real value, to the next smallest integer.

Example

INT(1.1) = 1
INT(-1.1) = -2

LN

Returns the natural log of a number.

Usage
    double = LN(number)
Description
    The number argument must be a positive real number. LN is the inverse function to EXP.
Example
    LN(1) = 0
    LN(2) = 0.693147
    LN(e) = 1
    LN(EXP(2)) = 2

Example

See Also

LOG , LOG10

LOG

Returns the logarithm of a number.

Usage

double = LOG(number, base)

Description

LOG takes:

number: any positive real number.

Base: the base of the log.

Example

LOG(10,10) = 1

LOG(3,2) = 1.584963

Example

See Also

LN , LOG10

LOG10

Returns the logarithm of a number.

Usage

double = LOG10(number)

Description

LOG10 takes any positive real number and returns a double.

Example

LOG10(10) = 1

LOG(3,2) = 1.584963

Example

See Also

LN , LOG

MDETERM

Returns a matrix determinate of an array.

Usage

double = MDETERM(string containing a number array)

Description

MDETERM calculates a determinate of an array. The array has to be passed as a string. The elements of each row need to be separated by commas, and each row separated by a semicolon. For example:

1	2
3	4

“1,2;3,4”

The matrix must be square or else an error occurs.

Example

MDETERM(“1,2;3,4”) = -2
MDETERM(“1.2,2.3;3.4,4.5”) = -2.42

Example

See Also

MMULT , PRODUCT

MMULT

Returns the product of two matrixes.

Usage

string = MMULT(string containing two number arrays)

Description

MMULT returns the product of two matrixes. The arrays have to be passed as a string. The elements of each row need to be separated by commas, and each row separated by a semicolon. Each matrix must be separated by curley brackets {} and a comma. For example:

1	2	6	7
3	4	8	9

“{1,2;3,4},{6,7;8,9}”
The number of columns in matrix 1 must be equal to the number of rows in matrix 2.
When calling the function via the DLL or Func-O-Matic, please note that the function declaration differs:

double **= MMULTVB(*safearray, safearray*)**
Visual Basic by default uses OLE based safearrays, so nothing special need be done. Simply pass two arrays of type double. In other languages you will have to construct such an array yourself. The method of doing this varies by language, and may product unpredictable results because both arrays and variant data types are implemented differently in different development environments.

double **= MMULT(*array of double, array of double, x1,y1,x2,y2*)**

If your language supports regular array types, you should use this form. It takes two arrays of type double. The x1 and y1 parameters are both 32 bit integers, and refer to the two dimensions of the array, likewise with x2 and y2.

Example

MMULT({1,2;3,4},{6,7;8,9}) = 22

Example

See Also

MDETERM , PRODUCT

MOD

Divides two numbers and returns only the remainder.

Usage

integer = MOD(number, divisor)

Description

MOD divides two numbers and returns only the remainder. The sign is the same as it would be in a normal division. Number and divisor should be integers.

MOD is equivalent to:

MOD(n, d) = n - d * INT(n / d)

Example

MOD(10,7) = 3
MOD(10,3) = 1
MOD(10,-3) = -2

Example

ODD

Rounds to the next higher odd integer, using absolute value.

Usage

integer = ODD(number)

Description

ODD rounds the number to the next odd number of larger absolute magnitude.

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:

integer = ODD(*double*)

Example

ODD(-2) = -3
ODD(1.2) = 3
ODD(4) = 5
ODD(1) = 1

Example

See Also

EVEN

PI

Returns the value of the mathematical constant PI.

Usage

number = PI()

Description

Returns the value of the mathematical constant PI to 15 digits.

Example

PI() = 3.14159265358979

Example

POWER

Returns a number raised to a given exponent.

Usage

double = POWER(number, exponent)

Description

POWER returns a number raised to a given exponent; numberexponent.

Example

POWER(2,3) = 8

Example

PRODUCT

Multiplies a series of numbers together.

Usage

number = PRODUCT(String of numbers)

Description

multiples PRODUCT takes a single string argument consisting of a series of numbers separated by commas, and those numbers together.

Example

PRODUCT("2,3,4.5") = 27

Example

See Also

MMULT , MDETERM

RADIANS

Converts degrees to radians.

Usage

number = RADIANS(Degrees)

Description

Translates an angle measurement from degrees to radians.
The formula is:

Radians = Degrees * PI / 180

Example

RADIANS(180) = PI
RADIANS(90) = PI / 2 = 1.570796

Example

See Also

DEGREES

RAND

Returns a random number between 0 and 1, not including 1.

Usage

double = RAND()

Description

Returns a random number between 0 and 1, not including 1. To make random numbers between different ranges, just multiply the RAND() result by the upper bounds.

Example

RAND() = random number 0   n < 1
RAND() * 2 = random number 0   n < 2
INT(RAND()*10) + 1 = random integer, 1   n   10

Example

ROMAN

Converts a number to a Roman numeral, as a string.

Usage

string = ROMAN(number, form)

Description

Converts a number to a Roman numberal, as a string. Roman numberals use the following letter to represent values:

I = 1
V = 5
X = 10
L = 50
C = 100
D = 500
M = 1000

Example

See Also

ARABIC

ROUND

Rounds a number to a specified significance.

Usage

number = ROUND(value, significance)

Description

Takes:

Value: any number.

Significance: Which digit to round to. Positive numbers mean the number of places to the right of the decimal. Zero means an integer. Negative values mean significant digits to the left of the decimal point.

Returns:

Number rounded up to the next higher absolute value is the significant digit is greater than or equal to five, and rounded down if it is less than 5.

Example

ROUND(133.666, 2) = 133.67
ROUND(133.333,1) = 133.3

Example

See Also

ROUNDDOWN , ROUNDUP , TRUNC

ROUNDDOWN

Rounds a number down to a specified significance.

Usage

number = ROUNDDOWN(value, significance)

Description

Takes:

Value: any number.

Significance: Which digit to round to. Positive numbers mean the number of places to the right of the decimal. Zero means an integer. Negative values mean significant digits to the left of the decimal point.

Returns:

Number always rounded down to the next lower significant digit.

Example

ROUNDDOWN(133.333,1) = 133.3
ROUNDDOWN(133.666,0) = 133

Example

See Also

ROUND , ROUNDUP , TRUNC

ROUNDUP

Rounds a number up to a specified significance.

Usage

number = ROUNDUP(value, significance)

Description

Takes:

Value: any number.

Significance: Which digit to round to. Positive numbers mean the number of places to the right of the decimal. Zero means an integer. Negative values mean significant digits to the left of the decimal point.

Returns:

Number always rounded up to the next higher significant digit.

Example

ROUNDUP(133.666, 2) = 133.67
ROUNDUP(133.333,1) = 133.4

Example

See Also

ROUND , ROUNDDOWN , TRUNC

SIGN

Returns only the sign of a number.

Usage

integer = SIGN(number)

Description

SIGN returns -1 for negative numbers, 0 for zero, and 1 for positive numbers. It's useful for dealing with things such as absolute values and roots where calculations might need to be done regardless of sign, then have the sign replaced.

Example

```
SIGN(10) = 1
SIGN(-10) = -1
SIGN(0) = 0
SIGN(-4) * SQRT(ABS(-4)) = -2
```

Example

SIN

Returns the sine of an angle.

Usage

number = SIN(angle)

Description

Returns a sine. The angle, by default, should be expressed in radians. To express in gradians or degrees, use the SetAngleMeasure function.

A sine is the ratio of the hypotenuse and the opposite sides of a right triangle. The ratio is always -1 < n < 1.

Example

SIN(0) = 0
SIN(0.5) = 0.479426

Example

See Also

COS , TAN

SINH

Returns the hyperbolic sine of a number.

Usage

double = SINH(number)

Description

Returns the hyperbolic sine of a number .
SINH is defined by:
SINH(x) = (ex - e-x) / 2

Example

SINH(7) = 548.3161

Example

See Also

COSH , TANH

SQRT

Returns a square root.

Usage

number = SQRT(number)

Description

SQRT return a square root. The number argument must be positive.

Example

SQRT(4) = 2

Example

See Also

SQR

SUM

Adds a series of numbers together.

Usage

number = SUM(String of numbers)

Description

SUM takes a single string argument consisting of a series of numbers separated by commas, and adds those numbers together.

Example

SUM(“2,3,4.5”) = 9.5

Example

See Also

SUMSQ , SUMX2MY2 , SUMX2PY2 , SUMXMY2

SUMSQ

Squares a series of numbers then adds the results together.

Usage

number = SUMSQ(String of numbers)

Description

SUMSQ takes a single string argument consisting of a series of numbers separated by commas. It squares each number then adds those results together.

Example

SUMSQ(“2,3,4.2”) = 30.64

NOTE:

When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

variant = SUMSQ(*safeArray*)

By default, Visual Basic implements arrays as OLE type SafeArray. Nothing special has to be done, simply pass the VB array. In other languages, various methods exist for making type SafeArray. Because both arrays and variant types are implemented differently in different development environments, care should be used with this function and it may not work correctly in your development environment.

Double = SUMSQ(*array of double, size*)

This version is preferable for those that can use it. It takes a standard array of double, and a single 32 bit integer value of how many dimensions exist in the array.

Example

See Also

SUM , SUMX2MY2 , SUMX2PY2 , SUMXMY2

SUMX2MY2

Returns the sum of the difference of squares .

Usage

number = SUMX2MY2(String of numbers, String of numbers)

Description

SUMX2MY2 takes 2 string arguments consisting of a series of numbers separated by commas, and each array surrounded by curly brackets {}. First it squares each element. Then it subtracts the elements from the second array from the elements of the first array. Finally it adds the numbers together.

Example

To get the sum of difference of the squares of two arrays, [2,3] and [4,2], do the following:

SUMX2MY2("{2,3},{4,2}") = -7
What happens is this: [2,3] and [4,2] gets squared. Then the elements from the first array get subtracted from the first - [4,9] - [16,4], or [-12,5]. Finally the elements of this array are summed, -12 + 5, or -7.

Example
See Also

SUM , SUMSQ , SUMX2PY2 , SUMXMY2

SUMX2PY2

Returns the sum of the sum of squares .

Usage

number = SUMX2PY2(String of numbers, String of numbers)

Description

SUMX2PY2 takes 2 string arguments consisting of a series of numbers separated by commas, and each array surrounded by curly brackets {}. First it squares each element. Then it adds all of the elements together.

Example

To get the sum of the sum of the squares of two arrays, [2,3] and [4,2], do the following:

SUMX2PY2("{2,3},{4,2}") = 33

Example

See Also

SUM , SUMSQ , SUMX2MY2 , SUMXMY2

SUMXMY2

Returns the sum of the squares of differences.

Usage

number = SUMXMY2(String of numbers, String of numbers)

Description

SUMXMY2 takes 2 string arguments consisting of a series of numbers separated by commas, and each array surrounded by curly brackets {}. First it subtracts elements of array 2 from array 1. Then it squares the results and finally adds the numbers together.

Example

SUMXMY2("{2,3},{4,2}") = 5

What happens is this: The elements of [2,3] and [4,2] gets subtracted resulting in a single array of [-2,1]. Then each element gets squared resulting in [4,1]. Finally each element is added together resulting in 5.

Example

See Also

SUM , SUMSQ , SUMX2MY2 , SUMX2PY2

TAN

Returns the tangent of an angle.

Usage

number = TAN(angle)

Description

Returns a tangent. The angle, by default, should be expressed in radians. To express in gradians or degrees, use the SetAngleMeasure function.

A tangent is the ratio of the opposite side and adjacent sides of a right triangle. The angle must always be -90 < angle < 90

Example

TAN(0) = 0
TAN(0.5) = 0.546302

Example

See Also

SIN , COS

TANH

Returns the hyperbolic tangent of a number.

Usage

double = TANH(number)

Description

Returns the hyperbolic tangent of a number .

TANH is defined by:
TANH(x) = SINH(x) / COSH(x) = (ex - e-x) / (ex + e-x)

Example

TANH(7) = 0.999998

Example

See Also

SINH , COSH

TRUNC

Rounds a number down to a specified significance.

Usage

number = TRUNC(value, significance)

Description

Takes:

Value: any number.

Significance: Which digit to round to. Positive numbers mean the number of places to the right of the decimal. Zero means an integer. Negative values mean significant digits to the left of the decimal point.

Returns:

Number always rounded down to the next lower significant digit.

Example

TRUNC(133.666, 2) = 133.66
TRUNC(133.333,1) = 133.3
TRUNC(133.666,0) = 133
TRUNC(133.666,-1) = 130
TRUNC(133.666,-2) = 100

Example

See Also

ROUND , ROUNDDOWN , ROUNDUP

DB

Returns the depreciation using fixed-declining balance method.

Usage

double = DB(cost, salvage, life, period, months)

Description

DB takes:

cost	The initial cost of an item.
Salvage	The value of the item at the end of the depreciation.
Life	How many years over which the item is depreciated.
Period	Which year the depreciation amount is being calculated for.
Months	Number of months in the first year of the item's life.

DB returns a double representing a currency amount of the depreciation value.

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used.
The variable types are:
double = DB(*double, double, integer, integer, integer*)

Example

DB(1000,100,10,3,5) = 149.52

Example

See Also

DDB , VDB

DDB

Returns the depreciation using double-declining balance method, or by a user supplied declining factor.

Usage

double = DDB(cost, salvage, life, period, factor)

Description

DDB takes:

cost	The initial cost of an item.
Salvage	The value of the item at the end of the depreciation.
Life	How many years over which the item is depreciated.
Period	Which year the depreciation amount is being calculated for.
Factor	Rate at which the balance declines. 2 means double.

DDB returns a double representing a currency amount of the depreciation value.

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:
double = DDB(*double, double, integer, integer, integer*)

Example

DDB(1000,100,10,3,2) = 128.00

Example

See Also

DB ,VDB

We want to hear from you!

- Is there a component you wish you could get, but can't find?
- Using a component that sucks and can't find a better one?

Let us know!

We want to hear what *real* people want and do our 110% best to supply it.

We're located at:

Component Café
PO Box 542269
Houston, TX 77254
1-888-889-5565

Check out our web-site often and you'll find:

- all the bug-fixes
- new products
- special offers

sample code and developer aids

WWW.COMPONENTCAFE.COM

WEBSITE

Check out our web-site often. All the bug-fixes, new products, and special offers can be found there.

HTTP://www.componentcafe.com

Installat
Upgrades and License Files
General License Agreement

EVAL_O_MA

FUNC_O_MA

CALC_O_MA

FV

Returns Future Value of an investment.

Usage

double = FV(interest rate, periods, payment, initial value, payment type)

Description

FV takes:

interest rate	The fixed interest rate. Careful, express the interest rate to the same measure as the periods. If periods are months, divide an annual interest rate by 12.
Periods	number of total payment periods.
Payment	amount of fixed payment made each period.
Initial value	the amount of the opening balance.
Payment type	“0” if payments are applied at the beginning of a period. “1” if payments are applied at the end of a period.

FV returns a double representing the currency value of the final value.

Signs of values should be watched with care. Negative values represent amounts going away from the person and into the account. Positive values mean towards the person and away from the investment. Thus a payment should be expressed as a negative number. A periodic draw off of the account should be positive number.

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:
double = FV(*double, integer, double, double, integer*)

Example

Given an initial deposit of $1000 with monthly deposits of $100 earning 10% annual interest for two years, FV would be calculated:

FV(0.10 / 12, 24, -100, -1000, 0) = 3865.08

Given an initial deposit of $10,000, monthly withdrawals of $100, a 10% interest rate and a two year period of time, FV would be:

FV(0.1/12, 24, 100, -10000, 0) = 9559.22

Example

See Also

PV , RATE , NPER , NPV

IPMT

Returns the amount of interest paid for a given period of a loan.

Usage

double = IPMT(interest rate, period, periods, initial value, final value, payment type)

Description

IPMT takes:

Interest rate	The fixed interest rate. Careful, express the interest rate to the same measure as the periods. If periods are months, divide an annual interest rate by 12.
Period	The period to calculate interest amount for.
Periods	Number of total payment periods.
Initial value	The amount of the opening balance.
Final Value	The amount of the final balance. Usually zero, but the final payment of a loan is often some residual amount.
Payment type	"0" if payments are applied at the beginning of a period. "1" if payments are applied at the end of a period.

IPMT returns a double representing the currency value of the amount of interest being paid for the specified period of the loan.

Signs of values should be watched with care. Negative values represent amounts going away from the person and into the account. Positive values mean towards the person and away from the loan. Thus a payment should be expressed as a negative number. A periodic draw off of the account should be positive number.

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:
double = IMPT(*double, integer, integer, double, double, integer*)

Example

For a loan of 10,000 at 10% interest that get's paid down to zero in 24 months, the amount of interest for the third month is:

IPMT(0.1/12,3,24,10000,0,0) = 77.01

Example

See Also

PMT , PPMT

NPER

Returns the number of periods of an investment.

Usage

double = NPER(interest rate, payment, current value, final value, payment type)

Description

NPER takes:

Interest rate	The fixed interest rate. Careful, express the interest rate to the same measure as the periods. If periods are months, divide an annual interest rate by 12.
Payment	Amount of fixed periodic payments
Current Value	The present of the investment or loan.
Final Value	The value the investment amount should finally attain, such as “0” for a loan that gets totally paid off.
Payment type	“0” if payments are applied at the beginning of a period. “1” if payments are applied at the end of a period.

Returns a double that tells how many periods payment has to be made to attain the final value. A fractional portion indicates a fractional last payment.

Signs of values should be watched with care. Negative values represent amounts going away from the person and into the account. Positive values mean towards the person and away from the loan. Thus a payment should be expressed as a negative number. A periodic draw off of the account should be positive number.

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:
double = NPER(*double, double, double, double, integer*)

Example

Given a loan of $10,000 with a 10% interest rate and $250 per month payments takes:

NPER(0.1/12, -250, 10000, 0,0) = 48.86 months to pay off.

Example

See Also

FV , PV , RATE , NPV

PMT

Returns the periodic payment of an annuity.

Usage

double = PMT(interest rate, periods, present value, future value, payment type)

Description

PMT takes:

Interest rate	The fixed interest rate. Careful, express the interest rate to the same measure as the periods. If periods are months, divide an annual interest rate by 12.
Periods	The number of periods payments are made.
Current Value	The present of the investment or loan.
Final Value	The value the investment amount should finally attain, such as “0” for a loan that gets totally paid off.
Payment type	“0” if payments are applied at the beginning of a period. “1” if payments are applied at the end of a period.

Returns a double representing the amount of each payment.

Signs of values should be watched with care. Negative values represent amounts going away from the person and into the account. Positive values mean towards the person and away from the loan. Thus a payment should be expressed as a negative number. A periodic draw off of the account should be positive number.

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:
double = PMT(*double, integer, double, double, integer*)

Example

Given a loan of $10,000 with a 10% interest rate, and a total of 24 monthly payments:
PMT(0.1/12, 24, 10000, 0,0) = -461.45 are the monthly payments.

Example

See Also

IPMT , PPMT

PPMT

Returns the amount of principle paid for a given period of a loan.

Usage

double = PPMT(interest rate, period, periods, initial value, final value, payment type)

Description

PPMT takes:

Interest rate	The fixed interest rate. Careful, express the interest rate to the same measure as the periods. If periods are months, divide an annual interest rate by 12.
Period	The period to calculate interest amount for.
Periods	Number of total payment periods.
Initial value	The amount of the opening balance.
Final Value	The amount of the final balance. Usually zero, but the final payment of a loan is often some residual amount.
Payment type	“0” if payments are applied at the beginning of a period. “1” if payments are applied at the end of a period.

PPMT returns a double representing the currency value of the amount of principle being paid for the specified period of the loan.

Signs of values should be watched with care. Negative values represent amounts going away from the person and into the account. Positive values mean towards the person and away from the loan. Thus a payment should be expressed as a negative number. A periodic draw off of the account should be positive number.

Example

For a loan of 10,000 at 10% interest that gets paid down to zero in 24 months, the amount of principle for the third month is:

PPMT(0.1/12,3,24,10000,0,0) = -384.44

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:
double = PPMT(*double, integer, integer, double, double, integer*)

Example

See Also

PMT , IPMT

PV

Returns Present Value of an investment.

Usage

double = PV(interest rate, periods, payment, future value, payment type)

Description

PV takes:

Interest rate	The fixed interest rate. Careful, express the interest rate to the same measure as the periods. If periods are months, divide an annual interest rate by 12.
Periods	number of total payment periods.
Payment	amount of fixed payment made each period.
Future value	the amount of the final value of the investment, such as 0 for a fully paid loan.
Payment type	“0” if payments are applied at the beginning of a period. “1” if payments are applied at the end of a period.

PV returns a double representing the currency value of the final value.

Signs of values should be watched with care. Negative values represent amounts going away from the person and into the account. Positive values mean towards the person and away from the investment. Thus a payment should be expressed as a negative number. A periodic draw off of the account should be positive number.

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:
double = PV(*double, integer, double, double, integer*)

Example

Given a loan where payments are $250 per month, that it gets paid off in 24 months, and that interest is 6%, what is the current value?

PV(0.06 / 12, 24, -250, 0, 0) = 5640.72

Example

See Also

FV , RATE , NPER , NPV

RATE

Determines the interest rate of an investment.

Usage

double = RATE(periods, payment, present value, future value, payment type, guess)

Description

RATE takes:

Periods	Number of periods payments are made.
Payment	amount of fixed payment made each period.
Present Value	Amount the investment is initially worth.
Future value	the amount of the final value of the investment, such as 0 for a fully paid loan.
Payment type	"0" if payments are applied at the beginning of a period. "1" if payments are applied at the end of a period.
Guess	Estimate of interest rate.

RATE returns a double representing approximate interest rate accurate to .00001.

Signs of values should be watched with care. Negative values represent amounts going away from the person and into the account. Positive values mean towards the person and away from the investment. Thus a payment should be expressed as a negative number. A periodic draw off of the account should be positive number.

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:
double = RATE(*integer, double, double, double, integer, double*)

Example

Given a loan with 36 payments, $300 payments, an initial value of 25,000 and a final value of 18,000, the interest rate is:

RATE(36, -300, 25000, -18000, 0.02) = 0.493, or 5.9% annually.

Example

See Also

FV , PV , NPER , NPV

SLN

Returns straight line depreciation.

Usage

double = SLN(cost, salvage, life)

Description

SLN takes:

cost	The initial cost of an item.
Salvage	The value of the item at the end of the depreciation.
Life	How many years over which the item is depreciated.

SLN returns a double representing a currency amount of the depreciation value.

Example

For an item costing $10,000, with depreciation life of 7 yearsm and a final value of $2000:

SLN(10000,2000,7) = 1142.86

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used.
The variable types are:
double = SLN(*double, double, integer*)

Example

See Also

SYD

SYD

Returns depreciation for a given period using sum-of-years digits method.

Usage

double = SYD(cost, salvage, life, period)

Description

SYD takes:

Cost	The initial cost of an item.
Salvage	The value of the item at the end of the depreciation.
Life	How many years over which the item is depreciated.
Period	Which year the depreciation amount is being calculated for.

SYD returns a double representing a currency amount of the depreciation value.

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used.
The variable types are:
double = SYD(*double, double, integer, integer*)

Example

For an item costing $10,000, with depreciation life of 3 years and a final value of $2000:

SYD(10000,2000,3,1) = 4000.00
SYD(10000,2000,3,2) = 2666.67
SYD(10000,2000,3,3) = 1333.33

Example

See Also

SLN

VDB

Returns the depreciation using a variable method.

Usage

double = VDB(cost, salvage, life, start period, end period, factor, method)

Description

VDB takes:

cost	The initial cost of an item.
Salvage	The value of the item at the end of the depreciation.
Life	How many years over which the item is depreciated.
Start Period	Starting period to calculate depreciation.
End Period	Ending period to calculate depreciation.
Factor	2 for double declining method. Number here applies same as the DDB method.
Method	True = use specified depreciation method throughout calculations. False = Switch to straight line depreciation when it's value is greater than the declining balance method.

VDB returns a double representing a currency amount of the depreciation value.

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:
double = VDB(*double, double, integer, integer, integer, integer, integer*)*

Example

VDB(1000,100,10,3,5,2,false) = 184.32

Example

See Also

DB , DDB

AVEDEV

Returns the Average Deviation of data points from their mean.

Usage
Double = AVEDEV(number1, number2….)
Description
Returns the average deviation of the arguments.
Example
AVEDEV(1,2,3,4,5) = 1.2
NOTE:
When calling the function via the DLL or Func-O-Matic, please be aware of the following differences:

***double* = AVEDEVVB(*SafeArray*)**
In Visual Basic the default type is SafeArray. In other languages you might have to set a variant variable to an array, or do other things to create a SafeArray type. Because each language handles arrays and OLE and variant datatypes differently, the function might not behave properly in other development environments.

***double* = AVEDEV(*array of double, integer*)**
If you're using a development environment that can take straight data arrays, you should use this syntax. It takes an array of double, and an integer that is the size of the array. It returns a double.

Example

MAX

Returns the maximum value out of a list of numbers.

Usage

number = MAX(number1, number2…)

Description

Returns the number with the highest value out of a list of numbers.

Example

MAX(1,2,3,4,5) = 5

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

variant = MAX(*safe array*)
In Visual Basic, the default array structure is of type SafeArray, and nothing special need be done. In other languages, you may need to set a variant variable equal to an array, or explicitly create a Safe Array datatype. Because different development environments handle arrays and variant data types differently, this function may not work properly in those environments.

double = MAX(*array of double, size*)
For other languages, you should use this syntax. It takes an array of double and an integer variable which is the size of the array.

Example

See Also

MIN

MEDIAN

Returns the median value of a list of numbers.

Usage

number = MEDIAN(number1, number2…)

Description

Returns the median of a list of numbers. The median is the number in the middle of a range - where half the values are higher and half the values are lower.

Example

MEDIAN(1,2,3,4,5) = 3
MEDIAN(1,9) = 5

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

variant = MEDIAN(*safe array*)
In Visual Basic, the default array structure is of type SafeArray, and nothing special need be done. In other languages, you may need to set a variant variable equal to an array, or explicitly create a Safe Array datatype. Because different development environments handle arrays and variant data types differently, this function may not work properly in those environments.

double = MEDIAN(*array of double, size*)
For other languages, you should use this syntax. It takes an array of double and an integer variable which is the size of the array.

Example

MIN

Returns the minimum value out of a list of numbers.

Usage
number = MIN(number1, number2…)

Description
Returns the number with the lowest value out of a list of numbers.

Example
MIN(1,2,3,4,5) = 5

Note:
When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

variant = MIN(*safe array*)
In Visual Basic, the default array structure is of type SafeArray, and nothing special need be done. In other languages, you may need to set a variant variable equal to an array, or explicitly create a Safe Array datatype. Because different development environments handle arrays and variant data types differently, this function may not work properly in those environments.

double = MIN(*array of double, size*)
For other languages, you should use this syntax. It takes an array of double and an integer variable which is the size of the array.

Example

See Also

MAX

MODE

Returns the mode value of a list of numbers.

Usage

number = MODE(number1, number2…)

Description

Returns the mode of a list of numbers. The mode is the number in a set of numbers that appears most frequently.

If there are no duplicate values, an error occurs.

Example

MODE(1,1,1,2,2) = 1
MODE(1,1,2,2) = 1
MODE(2,2,1,1) = 2

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

variant = MODE(*safe array*)
In Visual Basic, the default array structure is of type SafeArray, and nothing special need be done. In other languages, you may need to set a variant variable equal to an array, or explicitly create a Safe Array datatype. Because different development environments handle arrays and variant data types differently, this function may not work properly in those environments.

double = MODE(*array of double, size*)
For other languages, you should use this syntax. It takes an array of double and an integer variable which is the size of the array.

Example

PERMUT

Returns the number of permutations of permutations of objects you can form from a single set of objects.

Usage

integer = PERMUT(number of items total, number of items in a group)

Description

A permutation differs from a combination in that the order of items matters.

For example, if there were four colors, and you wanted to know how many permutations there were, you would have the following choices:

red + [blue, green, yellow]
blue + [red, green, yellow]
green + [red, blue, yellow]
yellow + [red, blue, green]

Which is 12 permutations. Notice that red + blue and blue + red are both counted as different. If order does not matter, you should use COMBIN instead.

Example

PERMUT(4,2) = 12
PERMUT(8,3) = 336

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:

integer = PERMUT(*integer, integer*)

Example

POISSON

Description
Returns the Poisson distribution. A common application of the Poisson distribution is predicting the number of events over a specific time, such as the number of cars arriving at a toll plaza in one minute.

Usage

Poisson(x, mean, cumulative)

X is the number of events.
Mean is the expected numeric value.
Cumulative is a logical value that determines the form of the probability distribution returned. If cumulative is TRUE, POISSON returns the cumulative Poisson probability that the number of random events occurring will be between zero and x inclusive; if FALSE, it returns the Poisson probability mass function that the number of events occurring will be exactly x.

If x is not an integer, it is truncated.
If x or mean is non-numeric, POISSON returns the #VALUE! error value.
If x £ 0, POISSON returns the #NUM! error value.
If mean £ 0, POISSON returns the #NUM! error value.
POISSON is calculated as follows.
For cumulative = FALSE:
For cumulative =TRUE:

Example
Poisson (2,5,FALSE) equals 0.084224
Poisson(2,5,TRUE) equals 0.124652.

Example

PROB

Description
Returns the probability that values in a range are between two limits. If upper_limit is not supplied, returns the probability that values in x_range are equal to lower_limit.

Usage
Prob(x_range, prob_range, lower_limit, upper_limit)

X_range	is the range of numeric values of x with which there are associated probabilities.
Prob_range	is a set of probabilities associated with values in x_range.
Lower_limit	is the lower bound on the value for which you want a probability.
Upper_limit	is the optional upper bound on the value for which you want a probability.

Example

Prob ({0,1,2,3},{0.2,0.3,0.1,0.4},2) equals 0.1
Prob({0,1,2,3},{0.2,0.3,0.1,0.4},1,3) equals 0.8.

Example

STDEV

Returns the standard deviation.

Usage

number = STDEV(number1, number2…)

Description

Returns the standard deviation of a list of numbers based on a sample of data.

Example

STDEV(1,2,3,4,5,6,7,8,9) = 2.738613

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

***variant* = STDEVVB(*safe array*)**

In Visual Basic, the default array structure is of type SafeArray, and nothing special need be done. In other languages, you may need to set a variant variable equal to an array, or explicitly create a Safe Array datatype. Because different development environments handle arrays and variant data types differently, this function may not work properly in those environments.

***double* = STDEV(*double, size*)**

For other languages, you should use this syntax. It takes an array of double and an integer variable which is the size of the array.

Example

See Also

STDEVP

STDEVP

Returns the standard deviation based on an entire population.

Usage

number = STDEVP(number1, number2…)

Description

Returns the standard deviation of a list of numbers based on a complete set of data.

Example

STDEVP(1,2,3,4,5,6,7,8,9) = 2.581989

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

***variant* = STDEVPVB(*safe array*)**

In Visual Basic, the default array structure is of type SafeArray, and nothing special need be done. In other languages, you may need to set a variant variable equal to an array, or explicitly create a Safe Array datatype. Because different development environments handle arrays and variant data types differently, this function may not work properly in those environments.

***double* = STDEVP(*double, size*)**

For other languages, you should use this syntax. It takes an array of double and an integer variable which is the size of the array.

Example

Stdevp(1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) equals 26.05

Example

See Also

STDEV

VAR

Estimates variance based on a sample of data.

Usage

number = VAR(number1, number2…)

Description

Returns the variance of a list of numbers based on a sample of the data.

Example

VAR(1,2,3,4,5,6,7,8,9) = 7.5

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

***variant* = VAR(*safe array*)**

In Visual Basic, the default array structure is of type SafeArray, and nothing special need be done. In other languages, you may need to set a variant variable equal to an array, or explicitly create a Safe Array datatype. Because different development environments handle arrays and variant data types differently, this function may not work properly in those environments.

***double* = VAR(*double, size*)**

For other languages, you should use this syntax. It takes an array of double and an integer variable which is the size of the array.

Example

Var(1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) equals 754.3.

Example

See Also

VARP

VARP

Estimates variance based on a complete set of data.

Usage
number = VARP(number1, number2…)
Description
Returns the variance of a list of numbers based on a complete set of the data.
Example
VAR(1,2,3,4,5,6,7,8,9) = 6.666667

Note:
When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

***variant* = VARP(*safe array*)**
In Visual Basic, the default array structure is of type SafeArray, and nothing special need be done. In other languages, you may need to set a variant variable equal to an array, or explicitly create a Safe Array datatype. Because different development environments handle arrays and variant data types differently, this function may not work properly in those environments.

***double* = VARP(*double, size*)**
For other languages, you should use this syntax. It takes an array of double and an integer variable which is the size of the array

Example
Varp (1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) equals 678.8.

Example

See Also

VAR

AND

Usage

AND(logical1, logical2, ...)
Logical1, logical2,...   are 1 to 30 conditions you want to test that can be either TRUE or FALSE.

The arguments should be logical values or arrays or references that contain logical values.

Description

Returns TRUE if all its arguments are TRUE; returns FALSE if one or more arguments is FALSE.

Example

AND(TRUE, TRUE) equals TRUE
AND(TRUE, FALSE) equals FALSE
AND(2+2=4, 2+3=5) equals TRUE

Example

See Also

NOT , OR

FALSE

Usage

FALSE( )

Description

Returns the logical value FALSE.

Example

See Also

TRUE

IF

Usage

IF(logical_test, value_if_true, value_if_false)

Logical_test   is any value or expression that can be evaluated to TRUE or FALSE.
Value_if_true   is the value that is returned if logical_test is TRUE. If logical_test is TRUE and value_if_true is omitted, TRUE is returned.
Value_if_false   is the value that is returned if logical_test is FALSE. If logical_test is FALSE and value_if_false is omitted, FALSE is returned.

Description
Returns one value if logical_test evaluates to TRUE and another value if it evaluates to FALSE.

Example

In the following example, if the value referred to by the name File is equal to "Chart", logical_test is TRUE and the macro function NEW(2) is carried out, otherwise, logical_test is FALSE and NEW(1) is carried out:
IF(File="Chart",NEW(2),NEW(1))

You could use the following nested IF function:

IF(Average>89,"A",IF(Average>79,"B",
IF(Average>69,"C",IF(Average>59,"D","F"))))

In the preceding example, the second IF statement is also the value_if_false argument to the first IF statement. Similarly, the third IF statement is the value_if_false argument to the second IF statement. For example, if the first logical_test (Average>89) is TRUE, "A" is returned. If the first logical_test is FALSE, the second IF statement is evaluated, and so on.General Description

Example

NOT

Usage

NOT(logical)

Logical is a value or expression that can be evaluated to TRUE or FALSE. If logical is FALSE, NOT returns TRUE; if logical is TRUE, NOT returns FALSE.

Description

Reverses the value of its argument. Use NOT when you want to make sure a value is not equal to one particular value.

Example

NOT(FALSE) equals TRUE
NOT(1+1=2) equals FALSE

Example

See Also

AND , OR

OR

Usage

OR(logical1, logical2, ...)

Logical1, logical2,... are 1 to 30 conditions you want to test that can be either TRUE or FALSE.

Description

Returns TRUE if any argument is TRUE; returns FALSE if all arguments are FALSE.

Example

OR(TRUE) equals TRUE
OR(1+1=1,2+2=5) equals FALSE

Example

See Also

AND , NOT

TRUE

Usage

TRUE( )

Description

Returns the logical value TRUE.

Example

Example

See Also

FALSE

CHAR

Returns the character specified by the code number.

Usage
Character = CHAR(integer)
Description
Reutrns the ANSI character set code for characters from 1 to 255.
Example
CHAR(65) = A

Example

CLEAN

Removes all non-printable characters from text.

Usage

string = CLEAN(string)

Description

CLEAN removes all non-printable characters from a string.

Example

CLEAN("¡hel" + Char(4) + "lo") = "hello"

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

***vbString* = CLEANVB(*vbString*)**

Because strings in Visual Basic are different from every one else, this is a special function for that string type.

String = CLEAN(*string*)

This takes and returns a standard C-style string which should be used by the languages that accept these.

Example

CODE

Returns the numeric code for a given character.

Usage
integer = CODE(string)
Description
Returns the number of the ANSI character set for the first character in the string.
Example
CODE("A") = 65

Note:
When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

integer = CODEVB(*vbString)*
Because strings in Visual Basic are different from every one else, this is a special function for that string type.

Integer = CODE(*string*)
This function takes a standard C-style string and should be used by most environments.

Example

See Also

MID , FIND , SEARCH

CONCATENATE

Joins several strings together into one larger string.

Usage

string = CONCATENATE(string1, string2…. String30)

Description

It joins up to 30 strings together.

Example

CONCATENATE("hello", " ", "there") = "hello there"

Example

DOLLAR

Takes a number and returns a number as a string formatted with commas and currency symbols based on Windows settings.

Usage

string = DOLLAR(number, decimals)

Description

Takes a number and returns a number as a string formatted with commas and currency symbols based on Windows settings. It trims to the number of decimals specified.

Example

DOLLAR(134.87,2) = "$134.87"
DOLLAR(1234.567,2) = "$1,234.56"

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

vbString = DOLLARVB(*double, integer*)
Because strings in Visual Basic are different from every one else, this is a special function for that string type.

String = DOLLAR(*double, integer*)
This function takes a standard C-style string and should be used by most environments.

Example

EXACT

Compares two strings, returns true if they're exactly the same, false if not.

Usage

result = EXACT(string1, string2)

Description

Compares two strings, returns true if they're exactly the same, false if not. Exact is case sensitive.

Example

```
EXACT("hello", "hello") = True
EXACT("Hello", "hello") = False
```

Example

FIND

Finds one string inside another and returns the position of the first character where it is found.

Usage

integer = FIND(string to find, main string, start position)

Description

Finds one string inside another and returns the position of the first character where it is found. It starts searching at the position specified in the third parameter, 0 starts at the beginning. FIND is case sensitive, while SEARCH is not.

Example

```
FIND("dog", "hotdog", 0) = 4
```

Example

See Also

MID , CODE , SEARCH

LEFT

Returns the left pert of a string.

Usage

string = LEFT(string, size)

Description

Returns the left part of a string. It returns only the number of characters specified by the size parameter.

Example

```
LEFT("hello", 2) = "he"
```

Example

See Also

RIGHT , MID

LEN

Returns the length of a string.

Usage

integer = LEN(String)

Description

Returns the length of a string.

Example

```
LEN("hello") = 5
```

Example

LOWER

Takes a string and returns the same string, but all letters are converted to lowercase.

Usage

Description

Takes a string and returns the same string, but all letters are converted to lowercase. Only upper case letters are changed.

Example

LOWER("Hello 17") = "hello 17"

NOTE:

When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

•1 Visual basic includes this function, so it is not necessary for VB string support.

String = LOWER(*string*)

This takes and returns a c-style string.

Example

See Also

UPPER , PROPER

MID

Returns a piece of a string.

Usage

string = MID(string, start pos, end pos)

Description

Returns a piece of a string, from the start and end positions specified. If the end position if bigger than the length of the string, then the whole string is returned.

Example

MID("hello", 2,3) = "ell"
MID("hello",2,10) = "ello

Example

See Also

CODE , FIND , SEARCH , LEFT , RIGHT

PROPER

Changes the capitalization of a string so that the first letter of each word is capitalized, and the rest of the letters are in lower case.

Usage

string = PROPER(string)

Description

Changes the capitalization of a string so that the first letter of each word is capitalized, and the rest of the letters are in lower case. It uses non-alphabet characters as separators.

Example

PROPER("hello there bob") = "Hello There Bob"
PROPER("joe's castle") = "Joe'S Castle"
PROPER("he9said") = "He9Said"
PROPER("MyBigFunctionName") = "Mybigfunctionname"

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

vbString = PROPERVB(*vbString*)
Because strings in Visual Basic are different from every one else, this is a special function for that string type.

String = PROPER(*string*)
This function takes and returns a standard C-Style string.

Example

See Also

LOWER , UPPER

REPLACE

Replaces part of one string with another.

Usage
string = REPLACE(original string, start position, size, new string)

Description
Replaces part of one string with another. It cuts out *size* number of characters starting at the start position, then inserts the new string in the same place.

Example
REPLACE("123456789",3,3,"abc") = "12abc6789"
REPLACE("123456789",3,1,"abc") = "12abc456789"

Example

See Also

SUBSTITUTE

REPT

Repeats a smaller string a number of times returning a larger string.

Usage

string = REPT(string, times)

Description

Repeats a smaller string a number of times returning a larger string.

Example

REPT("{}",10) = "{}{}{}{}{}{}{}{}{}{}"
REPT("joe",3) = "joejoejoe"

Example

RIGHT

Returns the right part of a string.

Usage
  string = RIGHT(string, size)
Description
  Returns the right part of a string. It returns only the number of characters specified by the size parameter.
Example
RIGHT("hello", 2) = "lo"

Example

See Also

LEFT , MID

SEARCH

Finds one string inside another and returns the position of the first character where it is found.

Usage

integer = Search(string to find, main string, start position)

Description

Finds one string inside another and returns the position of the first character where it is found. It starts searching at the position specified in the third parameter, 0 starts at the beginning. SEARCH is not case sensitive, while FIND is.

Example

SEARCH("dog", "HOTDOG", 0) = 4

Example

See Also

MID , CODE , FIND

SUBSTITUTE

Usage

SUBSTITUTE(text, old_text, new_text, instance_num)

Text is the text or the reference to a cell containing text for which you want to substitute characters.
Old_text is the text you want to replace.
New_text is the text you want to replace old_text with.
Instance_num specifies which occurrence of old_text you want to replace with new_text. If you specify instance_num, only that instance of old_text is replaced. Otherwise, every occurrence of old_text in text is changed to new_text.

Description
Substitutes new_text for old_text in a text string. Use SUBSTITUTE when you want to replace specific text in a text string; use REPLACE when you want to replace any text that occurs in a specific location in a text string.

Example

SUBSTITUTE("Sales Data", "Sales", "Cost") equals "Cost Data"
SUBSTITUTE("Quarter 1, 1991", "1", "2", 1) equals "Quarter 2, 1991"
SUBSTITUTE("Quarter 1, 1991", "1", "2", 3) equals "Quarter 1, 1992"

Example

See Also

REPLACE

TRIM

Usage

TRIM(text)

Text   is the text from which you want spaces removed.

Description
Removes all spaces from text except for single spaces between words. Use TRIM on text that you have received from another application that may have irregular spacing.

Example

TRIM(" First   Quarter   Earnings   ") equals "First Quarter Earnings"

Example

SEE ALSO
UPPER
safd';fdsa;lfdsa';fdsa
fdsa';fdsafdsa';fds

UPPER

Usage

UPPER(text)

Text is the text you want converted to uppercase. Text can be a reference or text string.

Description

Converts text to uppercase.

Example

UPPER("total") equals "TOTAL"

Example

See Also

LOWER , PROPER

CMFEET

Usage

CMFEET(number)
where number is the argument to be converted to feet.

Description

Converts Centimeter to Feet.

Example

CMFEET(30) gives 0.984252

Example

See Also

FEETCM , CMINCH

CMINCH

Usage

CMINCH(number)   where number is the argument to be converted to inch.

Description

Converts Centimeter to Inch.

Example

CMINCH(5) gives 1.9685

Example

See Also

CMFEET , FEETCM

CTOF

Converts from degrees centigrade to Fahrenheit.

Usage

number = CTOF(number)

Description

Takes an argument expressed as a temperature in centigrade, and returns the same measure converted to Fahrenheit.

Example

CTOF(100) = 212

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:

double = CTOF(*double*)

Example

CTOF(35) gives 95

Example

See Also

FTOC , CTOK , KTOC

CTOK

Usage

CTOK(number) where number is the argument to be converted to kelvin

Description

Converts Centigrade to Kelvin.

Example

CTOK(75) gives 167

Example

See Also

KTOC , CTOF , FTOC

FEETCM

Converts from feet to centimeters.

Usage

number = FEETCM(number)

Description

Takes an argument expressed in feet, and returns the same measure converted to centimeters.

Example

FEETCM(12) = 365.76

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:

double = FEETCM(*double*)

Example

FEETCM(1) gives 30.48

Example

See Also

CMFEET , CMINCH

FEETM

Usage

FEETM(number) where number is the argument to be converted to meter

Description

Converts Feet to Meter.

Example

FEETM(200) gives 60.96

Example

See Also

INCM , MFEET

FTOC

Usage

FTOC(number) where number is the number to converted to celcius

Description

Converts Farenheit to Celcius.

Example

FTOC(200) gives 93.33333

Example

See Also

CTOF , CTOK , KTOC

GALLTR

Converts from gallons to liters.

Usage

number = GALLTR(number)

Description

Takes an argument expressed in gallons, and returns the same measure converted to liters.

Example

GALLTR(1000) = 4546.09

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:

double = GALLTR(*double*)

Example

GALLTR(10) gives 45.4609

Example

See Also

LTRGAL

INCM

Usage

INCM(number) where number is the argument to be converted to centimeter

Description

Converts Inches to Centimeter

Example

INCM(20) gives 50.8

Example

See Also

FEETM , MFEET

KGPOUND

Usage

KGPOUND(number) is the argument to be converted to pound

Description

Converts Kilogram to Pound.

Example

KGPOUND(100) gives 220

Example

See Also

POUNDKG

KMMILE

Converts from kilometers to miles.

Usage
number = KMMILE(number)
Description
Takes an argument expressed in kilometers, and returns the same measure converted to miles.
Example
KMMILE(1000) = 621.38

Note:
When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:

double = KMMILE(*double*)

Example

KMMILE(8) gives 4.97104

Example

See Also

MILEKM

LTRGAL

Usage
LTRGAL(number) is the argument to be converted to gallons

Description
Converts Litre to Gallons.

Example

LTRGAL(100) gives 21.9969

Example

See Also

GALLTR

MFEET

Usage

MFEET(number) is the argument to be converted to feet

Description

Converts Meter to Feet.

Example

MFEET(12) gives 39.37007

Example

See Also

INCM , FEETM

MILEKM

Usage

MILEKM(number) is the argument to be converted to kilometer

Description

Converts Mile to Kilometer.

Example

MILEKM(100) gives 160.93

Example

See Also

KMMILE

MLOZ

Usage

MLOZ(number) where number is the argument to be converted to ounce

Description

Converts Milli to Ounce.

Example

MLOZ(202) gives 7.1104

Example

See Also

OZML

OZML

Converts from ounces to milliliters.

Usage
number = OZML(number)
Description
Takes an argument expressed in ounces, and returns the same measure converted to milliliters.
Example
OZML(300) = 8520

Note:
When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:

double = OZML(*double*)

Example

OZML(100) gives 2840

Example

See Also

MLOZ

POUNDKG

Usage

POUNDKG(number) where number is the argument to be converted to kilogram

Description

Converts Pounds to kilogram

Example

POUNDKG(20) gives 9.0909

Example

See Also

KGPOUND

SQFEETSQM

Usage

SQFEETSQM(number) where number is the number to be converted to squaremetre

Description

Converts Squarefeet to Squaremeter

Example

SQFEETSQM(100) gives 9.290304

Example

See Also

SQMSQFEET

SQMSQFEET

Usage
SQMSQFEET(number) where number is the argument to be converted to squarefeet
Description
Converts Squaremeter to Squarefeet.
Example

SQMSQFEET(10) gives 107.639

Example

See Also

SQFEETSQM

BITSLEFT

Takes a Long Integer and treats it as a string of 32 bits. It shifts the bits to the left.

Usage

Long Integer = BitsLeft(LongInteger, n)

Description

Takes a long integer and shifts the bits n bits to the left.

Example

if a variable **I** = 00000000 00000000 00000000 00001000 = 8 decimal,
then BitsLeft(**I**,2) = 00000000 00000000 00000000 00100000 = 32 decimal

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:

integer = BITSLEFT(*integer*, *integer*)

Example

See Also

BITSRIGHT , BITSOFF , BITSON

BITSRIGHT

Takes a Long Integer and treats it as a string of 32 bits. It shifts the bits to the right.

Usage
Long Integer = BitsRight(LongInteger, n)

Description
Takes a long integer and shifts the bits n bits to the right.

Example
if a variable I = 00000000 00000000 00000000 00001000 = 8 decimal,
then BitsRight(I,2) = 00000000 00000000 00000000 00000010 = 2 decimal

Note:
When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:

integer = BITSRIGHT(*integer, integer*)

Example

See Also

BITSLEFT , BITSOFF , BITSON

BITSOFF

Turns an individual bit off.

Usage
Long = BitsOff(Long, positon)

Description
Takes a long integer and treats it like a string of 32 bits. BitsOff turns the bit at the specified position to zero.

Example
if variable **I** = 00000000 00000000 00000000 11111111
then BitsOff(**I**,4) = 00000000 00000000 00000000 11110111

Note:
When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:

integer = BITSOFF(*integer, integer*)

Example

See Also

BITSLEFT, BITSRIGHT , BITSON

BITSON

Turns an individual bit off.

Usage

Long = BitsOn(Long, positon)

Description

Takes a long integer and treats it like a string of 32 bits. BitsOn turns the bit at the specified position to one.

Example

if variable **I** = 00000000 00000000 00000000 11111111
then BitsOff(**I**,9) = 00000000 00000000 00000001 11111111

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used. The variable types are:

integer = BITSON(*integer, integer*)

Example

See Also

BITSLEFT , BITSRIGHT , BITSOFF

COMPLEXADD

Adds two complex numbers together.

Usage

ComplexAdd(a1,b1,a2,b2,a3,b3)

Description

Adds two complex numbers together. Complex numbers are expressed in terms of their real and imaginary parts: a + ib. Where i is the square root of -1, a and b are both real numbers.

The function works like this:

a3 = (a1 + a2)
b3 = (b1 + b2)

Example

For two complex numbers: (2 + 3i) + (4+7i),
ComplexAdd(2,3,4,7,A,B);
A = 6
B=10

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used.
The variable types are:
double = COMPLEXADD(*double,double,double,double,double,double*)

Example

See Also

COMPLEXSUB , COMPLEXMULT

COMPLEXMULT

Multiplies two complex numbers together.

Usage

ComplexMult(a1,b1,a2,b2,a3,b3)

Description

Multiplies two complex numbers together. Complex numbers are expressed in terms of their real and imaginary parts: a + ib. Where i is the square root of -1, a and b are both real numbers.

The function works like this:

a3 + ib3 = (a1 + ib1) * (a2 - ib2) = ((a1*a2) + (b1*b2)) - ((a2*b1) - (a1*b2))i
a3 = ((a1*a2) + (b1*b2))
b3 = ((a2*b1) - (a1*b2))

Example

For two complex numbers: (2 + 3i) + (4+7i),
ComplexMult(2,3,4,7,A,B);
A = 29
B = -2

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used.
The variable types are:
double = COMPLEXMULT(*double,double,double,double,double,double*)

Example

See Also

COMPLEXADD , COMPLEXSUB

COMPLEXSUB

A Adds two complex numbers together.

Usage

ComplexSub(a1,b1,a2,b2,a3,b3)

Description

Subtracts two complex numbers together. Complex numbers are expressed in terms of their real and imaginary parts: a + ib. Where i is the square root of -1, a and b are both real numbers.

The function works like this:

a3 = (a1 - a2)
b3 = (b1 - b2)

Example

For two complex numbers: (2 + 3i) + (4+7i),
ComplexSub(2,3,4,7,A,B);
A = -2
B = -4

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variable types used.
The variable types are:
double = COMPLEXSUB(*double,double,double,double,double,double*)

Example

See Also

COMPLEXADD , COMPLEXMULT

SIN

Returns the sine of an angle.

Syntax

<OcxControl>.Sin(number)

The number argument can be any valid numeric expression that expresses an angle .

Note : Functions are not case sensitive.

Remarks

The Sin function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse.
The result lies in the range -1 to 1.

This example uses the Sin function to return the sine of an angle.

Examples

```
MyAngle = 1.3   ' Define angle in radians.
MyCosecant = 1 /<OcxControl>. Sin(MyAngle)    ' Calculate cosecant.
```

COS

Returns the cosine of an angle.

Syntax

<OcxControl>.Cos(number)

The number argument can be any valid numeric expression that expresses an angle in radians.

Note : Functions are not case sensitive.

Remarks

The Cos function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse.
The result lies in the range -1 to 1.
To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

This example uses the Cos function to return the cosine of an angle.

Examples

```
MyAngle = 1.3   ' Define angle in radians.
MySecant = 1 /<OcxControl>. Cos(MyAngle)      ' Calculate secant.
```

ACOS

Returns the arccosine of a number.

Syntax:

<OcxControl>.ACOS(number)

Number   is the cosine of the angle you want and must be from -1 to 1.
If you want to convert the result from radians to degrees, multiply it by 180/PI().

<u>Note : </u>Functions are not case sensitive.

Examples

<OcxControl>.ACOS(-0.5) equals 2.094395 (2p/3 radians)
<OcxControl>.ACOS(-0.5)*180/PI() equals 120 (degrees)

ASIN

Returns the arcsine of a number.

Syntax:

<OcxControl>.ASIN(number)
Number   is the sine of the angle you want and must be from -1 to 1.

Note : Functions are not case sensitive.

Remarks

To express the arcsine in degrees, multiply the result by 180/PI( ).

Examples

<OcxControl>.ASIN(-0.5) equals -0.5236 (-p/6 radians)
<OcxControl>.ASIN(-0.5)*180/PI() equals -30 (degrees)

ACOSH

Returns the inverse hyperbolic cosine of a number. Number must be greater than or equal to 1. The inverse hyperbolic cosine is the value whose hyperbolic cosine is number, so ACOSH(COSH(number)) equals number.

Syntax

<OcxControl>.ACOSH(number)

Number	is any real number equal to or greater than 1.

Note : Functions are not case sensitive.

Examples

<OcxControl>.ACOSH(1) equals 0
<OcxControl>.ACOSH(10) equals 2.993223

ASINH

Returns the inverse hyperbolic sine of a number. The inverse hyperbolic sine is the value whose hyperbolic sine is number, so ASINH(SINH(number)) equals number.

Syntax

<OcxControl>.ASINH(number)

Number	is any real number.

Note : Functions are not case sensitive.

Examples

<OcxControl>.ASINH(-2.5) equals -1.64723
<OcxControl>.ASINH(10) equals 2.998223

ATAN

Returns the arctangent of a number. The arctangent is the angle whose tangent is number. The returned angle is given in radians in the range -p/2 to p/2.

Syntax

<OcxControl>.ATAN(number)

Number   is the tangent of the angle you want.

Note : Functions are not case sensitive.

Remarks

To express the arctangent in degrees, multiply the result by 180/PI( ).

Examples

<OcxControl>.ATAN(1) equals 0.785398 (p/4 radians)
<OcxControl>.ATAN(1)*180/<OcxControl>.PI() equals 45 (degrees)

ATAN2

Returns the arctangent of the specified x- and y- coordinates. The arctangent is the angle from the x-axis to a line containing the origin (0, 0)) and a point with coordinates (x_num, y_num). The angle is given in radians between -p and p, excluding -p.

Syntax

<OcxControl>.ATAN2(x_num, y_num)

X_num	is the x-coordinate of the point.
Y_num	is the y-coordinate of the point.

Note : Functions are not case sensitive.

Remarks

A positive result represents a counterclockwise angle from the x-axis; a negative result represents a clockwise angle.

ATAN2(a,b) equals ATAN(b/a), except that a can equal 0 in ATAN2.
If both x_num and y_num are 0, ATAN2 returns the #DIV/0! error value.
To express the arctangent in degrees, multiply the result by 180/PI( ).

Examples

<OcxControl>.ATAN2(1, 1) equals 0.785398 (p/4 radians)
<OcxControl>.ATAN2(-1, -1) equals -2.35619 (-3p/4 radians)
<OcxControl>.ATAN2(-1, -1)*180/<OcxControl>.PI() equals -135 (degrees)

ATANH

Returns the inverse hyperbolic tangent of a number. Number must be between -1 and 1 (excluding -1 and 1). The inverse hyperbolic tangent is the value whose hyperbolic tangent is number, so ATANH(TANH(number)) equals number.

Syntax

<OcxControl>.ATANH(number)

Number	is any real number between 1 and -1.

Note : Functions are not case sensitive.

Examples

<OcxControl>.ATANH(0.76159416) equals 1, approximately
<OcxControl>.ATANH(-0.1) equals -0.10034

COSH

Returns the hyperbolic cosine of a number.

Syntax

<OcxControl>.COSH(number)

The formula for the hyperbolic cosine is:

Note : Functions are not case sensitive.

Examples

<OcxControl>.COSH(4) equals 27.30823
<OcxControl>.COSH(<OcxControl>.EXP(1)) equals 7.610125, where EXP(1) is e, the base of the natural logarithm.

SINH

Returns the hyperbolic sine of a number.

Syntax

<OcxControl>.SINH(number)

Number   is any real number.

The formula for the hyperbolic sine is:

Note : Functions are not case sensitive.

Examples

<OcxControl>.SINH(1) equals 1.175201194
<OcxControl>.SINH(-1) equals -1.175201194

You can use the hyperbolic sine function to approximate a cumulative probability distribution. Suppose a laboratory test value varies between 0 and 10 seconds. An empirical analysis of the collected history of experiments shows that the probability of obtaining a result, x, of less than t seconds is approximated by the following equation:
P(x<t) = 2.868 * SINH(0.0342 * t), where 0<t<10
To calculate the probability of obtaining a result of less than 1.03 seconds, substitute 1.03 for t:

2.868*SINH(0.0342*1.03) equals 0.101049063
You can expect this result to occur about 101 times for every 1000 experiments.

TAN

Returns the tangent of the given angle.

Syntax

<OcxControl>.TAN(number)

Number is the angle in radians for which you want the tangent. If your argument is in degrees, multiply it by PI()/180 to convert it to radians.

Note : Functions are not case sensitive.

Examples

<OcxControl>.TAN(0.785) equals 0.99920
<OcxControl>.TAN(45*PI()/180) equals 1

TANH

Returns the hyperbolic tangent of a number.

Syntax

<OcxControl>.TANH(number)
Number is any real number

The formula for the hyperbolic tangent is:

Note : Functions are not case sensitive.

Examples

<OcxControl>.TANH(-2) equals -0.96403
<OcxControl>.TANH(0) equals 0
<OcxControl>.TANH(0.5) equals 0.462117

ABS

Returns the absolute value of a number. The absolute value of a number is the number without its sign.

Syntax

<OcxControl>.ABS(number)

Number   is the real number of which you want the absolute value.

Note : Functions are not case sensitive.

Examples

<OcxControl>.ABS(2) equals 2
<OcxControl>.ABS(-2) equals 2

CEILING

Returns number rounded up, away from zero, to the nearest multiple of significance. For example, if you want to avoid using pennies in your prices and your product is priced at $4.42, use the formula =CEILING(4.42,0.05) to round prices up to the nearest nickel.

Syntax

<OcxControl>.CEILING(number, significance)

Number	is the value you want to round.
Significance	is the multiple to which you want to round.

Note : Functions are not case sensitive.

Remarks

If either argument is non-numeric, CEILING returns the #VALUE! error value.
Regardless of the sign of number, a value is rounded up when adjusted away from zero. If number is an exact multiple of significance, no rounding occurs.
If number and significance have different signs, CEILING returns the #NUM! error value.

Examples

<OcxControl>.CEILING(2.5, 1) equals 3
<OcxControl>.CEILING(-2.5, -2) equals -4
<OcxControl>.CEILING(-2.5, 2) equals #NUM!
<OcxControl>.CEILING(1.5, 0).1) equals 1.5
<OcxControl>.CEILING(0.234, 0).01) equals 0.24

FLOOR

Rounds number down, toward zero, to the nearest multiple of significance.

Syntax

<OcxControl>.FLOOR(number, significance)

Number	is the numeric value you want to round.
Significance	is the multiple to which you want to round.

Note : Functions are not case sensitive.

Remarks

If either argument is non-numeric, FLOOR returns the #VALUE! error value.

If number and significance have different signs, FLOOR returns the #NUM! error value.

Regardless of the sign of number, a value is rounded down when adjusted away from zero. If number is an exact multiple of significance, no rounding occurs.

Examples

<OcxControl>.FLOOR(2.5, 1) equals 2
<OcxControl>.FLOOR(-2.5, -2) equals -2
<OcxControl>.FLOOR(-2.5, 2) equals #NUM!
<OcxControl>.FLOOR(1.5, 0).1) equals 1.5
<OcxControl>.FLOOR(0.234, 0).01) equals 0.23

COMBIN

Returns the number of combinations for a given number of objects. Use COMBIN to determine the total possible number of groups for a given number of objects.

Syntax

<OcxControl>.COMBIN(number, number_chosen)

Number is the number of objects.
Number_chosen is the number of objects in each combination.

Note : Functions are not case sensitive.

Remarks

Numeric arguments are truncated to integers.
If either argument is non-numeric, COMBIN returns the #NAME? error value.
If number < 0, number_chosen < 0, or number < number_chosen, COMBIN returns the #NUM! error value.
A combination is any set or subset of objects, regardless of their internal order. Combinations are distinct from permutations, for which the internal order is significant.
The number of combinations is as follows, where number = n and number_chosen = k:

where:

Example

Suppose you want to form a two-person team from eight candidates and you want to know how many possible teams can be formed.

<OcxControl>. COMBIN(8, 2) equals 28 teams.

DEGREES

Converts radians into degrees.

Syntax

<OcxControl>.DEGREES(angle)

Angle is the angle in radians that you want to convert.

Note : Functions are not case sensitive.

Example

<OcxControl>.DEGREES(<OcxControl>.PI()) equals 180

RADIANS

Converts degrees to radians.

Syntax

<OcxControl>.RADIANS(angle)

Angle is an angle in degrees that you want to convert.

Note : Functions are not case sensitive.

Example

<OcxControl>.RADIANS(270) equals 4.712389 (3p/2 radians)

EVEN

Returns number rounded up to the nearest even integer. You can use this function for processing items that come in twos. For example, a packing crate accepts rows of one or two items. The crate is full when the number of items, rounded up to the nearest two, matches the crate's capacity.

Syntax

<OcxControl>.EVEN(number)

Number is the value to round.

Note : Functions are not case sensitive.

Remarks

If number is non-numeric, EVEN returns the #VALUE! error value.
Regardless of the sign of number, a value is rounded up when adjusted away from zero. If number is an even integer, no rounding occurs.

Examples

<OcxControl>.EVEN(1.5) equals 2
<OcxControl>.EVEN(3) equals 4
<OcxControl>.EVEN(2) equals 2
<OcxControl>.EVEN(-1) equals -2

ODD

Returns number rounded up to the nearest odd integer.

Syntax

<OcxControl>.ODD(number)

Number	is the value to round.

Note : Functions are not case sensitive.

Remarks

If number is non-numeric, ODD returns the #VALUE! error value.
Regardless of the sign of number, a value is rounded up when adjusted away from zero. If number is an odd integer, no rounding occurs.

Examples

<OcxControl>.ODD(1.5) equals 3
<OcxControl>.ODD(3) equals 3
<OcxControl>.ODD(2) equals 3
<OcxControl>.ODD(-1) equals -1
<OcxControl>.ODD(-2) equals -3

SUM

Returns the sum of all the numbers in the list of arguments.

Syntax

<OcxControl>.SUM(number1, number2, ...)

Number1, number2,...	are 1 to 30 arguments for which you want the sum.

Numbers, logical values, and text representations of numbers that you type directly into the list of arguments are counted. See the first and second examples following.

Arguments that are error values or text that cannot be translated into numbers cause errors.

Note : Functions are not case sensitive.

Examples

<OcxControl>.SUM(3, 2) equals 5
<OcxControl>.SUM("3", 2, TRUE) equals 6 because the text values are translated into numbers, and the logical value TRUE is translated into the number 1.

SUMSQ

Returns the sum of the squares of the arguments.

Syntax

<OcxControl>.SUMSQ(number1, number2, ...)

Number1, number2,...   are 1 to 30 arguments for which you want the sum of the squares. You can also use a single array or a reference to an array instead of arguments separated by commas.

Note : Functions are not case sensitive.

Example

<OcxControl>.SUMSQ(3, 4) equals 25.

SUMX2MY2

Returns the sum of the difference of squares of corresponding values in two arrays.

Syntax

<OcxControl>.SUMX2MY2(array_x, array_y)

Array_x	is the first array or range of values.
Array_y	is the second array or range of values.

Note : Functions are not case sensitive.

Remarks

The arguments should be numbers, or names, arrays, or references that contain numbers.

If array_x and array_y have a different number of values, SUMX2MY2 returns the #N/A error value.

The equation for the sum of the difference of squares is:

Example

<OcxControl>.SUMX2MY2({2, 3, 9, 1, 8, 7, 5}, {6, 5, 11, 7, 5, 4, 4}) equals -55.

SUMX2PY2

Returns the sum of the sum of squares of corresponding values in two arrays. The sum of the sum of squares is a common term in many statistical calculations.

Syntax

<OcxControl>.SUMX2PY2(array_x, array_y)

Array_x	is the first array or range of values.
Array_y	is the second array or range of values.

Note : Functions are not case sensitive.

Remarks

The arguments should be numbers, or names, arrays, or references that contain numbers.
If array_x and array_y have a different number of values, SUMX2PY2 returns the #N/A error value.
The equation for the sum of the sum of squares is:

Example

<OcxControl>.SUMX2PY2({2, 3, 9, 1, 8, 7, 5}, {6, 5, 11, 7, 5, 4, 4}) equals 521.

SUMXMY2

Returns the sum of squares of differences of corresponding values in two arrays.

Syntax

<OcxControl>.SUMXMY2(array_x, array_y)

Array_x	is the first array or range of values.
Array_y	is the second array or range of values.

Note : Functions are not case sensitive.

Remarks

The arguments should be numbers, or names, arrays, or references that contain numbers.
If array_x and array_y have a different number of values, SUMXMY2 returns the #N/A error value.
The equation for the sum of squared differences is:

Example

<OcxControl>.SUMXMY2({2, 3, 9, 1, 8, 7, 5}, {6, 5, 11, 7, 5, 4, 4}) equals 79.

ROUND

Rounds a number to a specified number of digits.

Syntax

<OcxControl>.ROUND(number, num_digits)

Number	is the number you want to round.
Num_digits	specifies the number of digits to which you want to round number.

If num_digits is greater than 0, then number is rounded to the specified number of decimal places.
If num_digits is 0, then number is rounded to the nearest integer.
If num_digits is less than 0, then number is rounded to the left of the decimal point.

Note : Functions are not case sensitive.

Examples

<OcxControl>.ROUND(2.15, 1) equals 2.2
<OcxControl>.ROUND(2.149, 1) equals 2.1
<OcxControl>.ROUND(-1.475, 2) equals -1.48
<OcxControl>.ROUND(21.5, -1) equals 20.

ROUNDDOWN

Rounds a number down, toward zero.

Syntax

<OcxControl>.ROUNDDOWN(number, num_digits)
Number is any real number that you want rounded down.
Num_digits is the number of digits to which you want to round number.

Note : Functions are not case sensitive.

Remark

ROUNDDOWN behaves like ROUND, except that it always rounds a number down.
If num_digits is greater than 0, then number is rounded down to the specified number of decimal places.
If num_digits is 0 or omitted, then number is rounded down to the nearest integer.
If num_digits is less than 0, then number is rounded down to the left of the decimal point.

Examples

<OcxControl>.ROUNDDOWN(3.2, 0)) equals 3
<OcxControl>.ROUNDDOWN(76.9,0) equals 76
<OcxControl>.ROUNDDOWN(3.14159, 3) equals 3.141
<OcxControl>.ROUNDDOWN(-3.14159, 1) equals -3.1
<OcxControl>.ROUNDDOWN(31415.92654, -2) equals 31400.

ROUNDUP

Rounds a number up, away from zero.

Syntax

<OcxControl>.ROUNDUP(number, num_digits)

Number	is any real number that you want rounded up.
Num_digits	is the number of digits to which you want to round number.

Note : Functions are not case sensitive.

Remarks

ROUNDUP behaves like ROUND, except that it always rounds a number up.

If num_digits is greater than 0, then number is rounded up to the specified number of decimal places.

If num_digits is 0 or omitted, then number is rounded up to the nearest integer.

If num_digits is less than 0, then number is rounded up to the left of the decimal point.

Examples

<OcxControl>.ROUNDUP(3.2,0) equals 4
<OcxControl>.ROUNDUP(76.9,0) equals 77
<OcxControl>.ROUNDUP(3.14159, 3) equals 3.142
<OcxControl>.ROUNDUP(-3.14159, 1) equals -3.2
<OcxControl>.ROUNDUP(31415.92654, -2) equals 31500.

TRUNC

Truncates a number to an integer by removing the fractional part of the number.

Syntax

<OcxControl>.TRUNC(number, num_digits)

Number	is the number you want to truncate.
Num_digits	is a number specifying the precision of the truncation. The default value for num_digits is zero.

Note : Functions are not case sensitive.

Remarks

TRUNC and INT are similar in that both return integers. TRUNC removes the fractional part of the number. INT rounds numbers down to the nearest integer based on the value of the fractional part of the number. INT and TRUNC are different only when using negative numbers: TRUNC(-4.3) returns -4, but INT(-4.3) returns -5, because -5 is the lower number.

Examples

<OcxControl>.TRUNC(8.9) equals 8
<OcxControl>.TRUNC(-8.9) equals -8
<OcxControl>.TRUNC(PI()) equals 3.

LN

Returns the natural logarithm of a number. Natural logarithms are based on the constant e (2.71828182845904).

Syntax

<OcxControl>.LN(number)

Number	is the positive real number for which you want the natural logarithm.

Note : Functions are not case sensitive.

Remarks

LN is the inverse of the EXP function.

Examples

<OcxControl>.LN(86) equals 4.454347
<OcxControl>.LN(2.7182818) equals 1
<OcxControl>.LN(EXP(3)) equals 3
<OcxControl>.EXP(<OcxControl>.LN(4)) equals 4.

LOG

Returns the logarithm of a number to the base you specify.

Syntax

<OcxControl>.LOG(number, base)

Number	is the positive real number for which you want the logarithm.
Base	is the base of the logarithm. If base is omitted, it is assumed to be 10.

Note : Functions are not case sensitive.

Examples

<OcxControl>.LOG(10) equals 1
<OcxControl>.LOG(8, 2) equals 3
<OcxControl>.LOG(86, 2.7182818) equals 4.454347.

LOG10

Returns the base-10 logarithm of a number.

Syntax

<OcxControl>.LOG10(number)

Number	is the positive real number for which you want the base-10 logarithm.

Note : Functions are not case sensitive.

Examples

<OcxControl>.LOG10(86) equals 1.934498451
<OcxControl>.LOG10(10) equals 1
<OcxControl>.LOG10(1E5) equals 5
<OcxControl>.LOG10(10^5) equals 5.

EXP

Returns e raised to the power of number. The constant e equals 2.71828182845904, the base of the natural logarithm.

Syntax

<OcxControl>.EXP(number)

Number is the exponent applied to the base e.

Note : Functions are not case sensitive.

Remarks

To calculate powers of other bases, use the exponentiation operator (^).
EXP is the inverse of LN, the natural logarithm of number.

Examples

<OcxControl>.EXP(1) equals 2.718282 (the approximate value of e)
<OcxControl>.EXP(2) equals e2, or 7.389056
<OcxControl>.EXP(<OcxControl>.LN(3)) equals 3.

FACT

Returns the factorial of a number. The factorial of a number is equal to 1*2*3*...* number.

Syntax

<OcxControl>.FACT(number)

Number is the nonnegative number you want the factorial of. If number is not an integer, it is truncated.

Note : Functions are not case sensitive.

Examples

<OcxControl>.FACT(1) equals 1
<OcxControl>.FACT(1.9) equals FACT(1) equals 1
<OcxControl>.FACT(0) equals 1
<OcxControl>.FACT(-1) equals #NUM!
<OcxControl>.FACT(5) equals 1*2*3*4*5 equals 120.

FIX

Rounds a number down to the nearest integer.

Syntax

<OcxControl>.FIX(number)

Number	is the real number you want to round down to an integer.

Note : Functions are not case sensitive.

Examples

<OcxControl>.FIX(8.9) equals 8
<OcxControl>.FIX(-8.9) equals -9

MDETERM

Returns the matrix determinant of an array.

Syntax

<OcxControl>.MDETERM(array)

Array is a numeric array with an equal number of rows and columns.

such as {1,2,3;4,5,6;7,8,9}; or as a name to either of these.

Note : Functions are not case sensitive.

Remarks

The matrix determinant is a number derived from the values in array.

Examples

<OcxControl>.MDETERM({1,3,8,5;1,3,6,1;1,1,1,0;7,3,10,2}) equals 88
<OcxControl>.MDETERM({3,6,1;1,1,0;3,10,2}) equals 1
<OcxControl>.MDETERM({3,6;1,1}) equals -3
MDETERM({1,3,8,5;1,3,6,1}) equals #VALUE! because the array does not have an equal number of rows and columns.

MMULT

Returns the matrix product of two arrays. The result is an array with the same number of rows as array1 and the same number of columns as array2.

Syntax

<OcxControl>.MMULT(array1, array2)

Array1, array2	are the arrays you want to multiply.

The number of columns in array1 must be the same as the number of rows in array2, and both arrays must contain only numbers.

Note : Functions are not case sensitive.

Remarks
The matrix product array a of two arrays b and c is:
where i is the row number and j is the column number.
Formulas that return arrays must be entered as array formulas.

Examples

<OcxControl>.MMULT({1,3;7,2}, {2,0;0,2}) equals {2,6;14,4}
<OcxControl>.MMULT({3,0;2,0}, {2,0;0,2}) equals {6,0;4,0}
<OcxControl>.MMULT({1,3,0;7,2,0;1,0,0}, {2,0;0,2}) equals #VALUE!, because the first array has three columns and the second array has only two rows.

PRODUCT

Multiplies all the numbers given as arguments and returns the product.

Syntax

<OcxControl>.PRODUCT(number1, number2, ...)

Number1, number2,...   are 1 to 30 numbers that you want to multiply.

Note : Functions are not case sensitive.

Remarks

Arguments that are numbers, logical values, or text representations of numbers are counted; arguments that are error values or text that cannot be translated into numbers cause errors.

If an argument is an array or reference, only numbers in the array or reference are counted.

Examples

<OcxControl>.PRODUCT(5:15) equals 2250

MOD

Returns the remainder after number is divided by divisor. The result has the same sign as divisor.

Syntax

<OcxControl>.MOD(number, divisor)

Number   is the number for which you want to find the remainder.
Divisor   is the number by which you want to divide number. If divisor is 0, MOD returns the #DIV/0! error value.

Note : Functions are not case sensitive.

Remarks

The MOD function can be expressed in terms of the INT function:

<OcxControl>.MOD(n, d) = n - d*<OcxControl>.INT(n/d)
Examples

<OcxControl>.MOD(3, 2) equals 1
<OcxControl>.MOD(-3, 2) equals 1
<OcxControl>.MOD(3, -2) equals -1
<OcxControl>.MOD(-3, -2) equals -1.

PI

Remarks

Returns the value of pi.

Note : Functions are not case sensitive.

This function does not take any arguments.

POWER

Returns the result of a number raised to a power.

Syntax

<OcxControl>.POWER(number, power)
Number   is the base number.   It can be any real number.
Power   is the exponent, to which the base number is raised.

Note : Functions are not case sensitive.

Remark
The "^" operator can be used instead of POWER to indicate to what power the base number is to be raised, such as in 5^2

Examples

<OcxControl>.POWER(5,2) equals 25
<OcxControl>.POWER(98.6,3.2) equals 2401077
<OcxControl>.POWER(4,5/4) equals 5.656854.

RAND

Returns an evenly distributed random number greater than or equal to 0 and less than 1. A new random number is returned every time the worksheet is calculated.

Syntax

<OcxControl>.RAND( )

Note : Functions are not case sensitive.

Remarks
To generate a random real number between a and b, use:

<OcxControl>.RAND()*(b-a)+a

Examples
To generate a random number greater than or equal to 0 but less than 100:

<OcxControl>.RAND()*100.

ROMAN

Converts an Arabic numeral to Roman, as text.

Syntax

<OcxControl>.ROMAN(number, form)

Number	is the Arabic numeral you want converted.
Form	is a number specifying the type of Roman numeral you want. The Roman numeral style ranges from Classic to Simplified, becoming more concise as the value of form increases. See the example following ROMAN(499,0) below.

Form	Type
0 or omitted	Classic
1	More concise. See example below
2	More concise. See example below
3	More concise. See example below
4	Simplified
TRUE	Classic
FALSE	Simplified

Note : Functions are not case sensitive.

Remarks

If number is negative the #value! error value is returned.

If number is greater than 3999, the #value error value is returned.

Examples

<OcxControl>.ROMAN(499,0) equals "CDXCIX"
<OcxControl>.ROMAN(499,1) equals "LDVLIV"
<OcxControl>.ROMAN(499,2) equals "XDIX"
<OcxControl>.ROMAN(499,3) equals "VDIV"
<OcxControl>.ROMAN(499,4) equals "ID"
<OcxControl>.ROMAN(1993,0) equals "MCMXCIII".

SIGN

Determines the sign of a number. Returns 1 if number is positive, 0) if number is 0, and -1 if number is negative.

Syntax

<OcxControl>.SIGN(number)
Number   is any real number.

Note : Functions are not case sensitive.

Examples

<OcxControl>.SIGN(10) equals 1
<OcxControl>.SIGN(4-4) equals 0
<OcxControl>.SIGN(-0.00001) equals -1.

SQRT

Returns a positive square root.

Syntax

<OcxControl>.SQRT(number)

Number is the number for which you want the square root. If number is negative, SQRT returns the #NUM! error value.

Note : Functions are not case sensitive.

Examples

<OcxControl>.SQRT(16) equals 4
<OcxControl>.SQRT(-16) equals #NUM!
<OcxControl>.SQRT(<OcxControl>.ABS(-16)) equals 4.

DB

Returns the depreciation of an asset for a specified period using the fixed-declining balance method.

Syntax

<OcxControl>.DB(cost, salvage, life, period, month)

Cost is the initial cost of the asset.
Salvage is the value at the end of the depreciation (sometimes called the salvage value of the asset).
Life is the number of periods over which the asset is being depreciated (sometimes called the useful life of the asset).
Period is the period for which you want to calculate the depreciation. Period must use the same units as life.
Month is the number of months in the first year. If month is omitted, it is assumed to be 12.

Note : Functions are not case sensitive.

Remarks
The fixed-declining balance method computes depreciation at a fixed rate. DB uses the following formulas to calculate depreciation for a period:
(cost - total depreciation from prior periods) * rate
where:
rate = 1 - ((salvage / cost) ^ (1 / life)), rounded to three decimal places
Depreciation for the first and last periods are special cases. For the first period, DB uses this formula:
cost * rate * month / 12
For the last period, DB uses this formula:
((cost - total depreciation from prior periods) * rate * (12 - month)) / 12

Examples

Suppose a factory purchases a new machine. The machine costs $1,000,000 and has a lifetime of six years. The salvage value of the machine is $100,000. The following examples show depreciation over the life of the machine. The results are rounded to whole numbers.
<OcxControl>.DB(1000000,100000,6,1,7) equals $186,083
<OcxControl>.DB(1000000,100000,6,2,7) equals $259,639
<OcxControl>.DB(1000000,100000,6,3,7) equals $176,814
<OcxControl>.DB(1000000,100000,6,4,7) equals $120,411
<OcxControl>.DB(1000000,100000,6,5,7) equals $82,000
<OcxControl>.DB(1000000,100000,6,6,7) equals $55,842
<OcxControl>.DB(1000000,100000,6,7,7) equals $15,845.

DDB

Returns the depreciation of an asset for a spcified period using the double-declining balance method or some other method you specify.

Syntax

<OcxControl>.DDB(cost, salvage, life, period, factor)

Cost is the initial cost of the asset.
Salvage is the value at the end of the depreciation (sometimes called the salvage value of the asset).
Life is the number of periods over which the asset is being depreciated (sometimes called the useful life of the asset).
Period is the period for which you want to calculate the depreciation. Period must use the same units as life.
Factor is the rate at which the balance declines. If factor is omitted, it is assumed to be 2 (the double-declining balance method).

All five arguments must be positive numbers.

Note : Functions are not case sensitive.

Remarks
The double-declining balance method computes depreciation at an accelerated rate. Depreciation is highest in the first period and decreases in successive periods. DDB uses the following formula to calculate depreciation for a period:
cost - salvage(total depreciation from prior periods) * factor / life
Change factor if you do not want to use the double-declining balance method.

Examples

Suppose a factory purchases a new machine. The machine costs $2400 and has a lifetime of 10 years. The salvage value of the machine is $300. The following examples show depreciation over several periods. The results are rounded to two decimal places.
<OcxControl>.DDB(2400,300,3650,1) equals $1.32, the first day's depreciation.
<OcxControl>.DDB(2400,300,120,1,2) equals $40.00, the first month's depreciation.
<OcxControl>.DDB(2400,300,10,1,2) equals $480.00, the first year's depreciation.
<OcxControl>.DDB(2400,300,10,2,1.5) equals $306.00, the second year's depreciation using a factor of 1.5 instead of the double-declining balance method.

VDB

Returns the depreciation of an asset for any period you specify, including partial periods, using the double-declining balance method or some other method you specify. VDB stands for variable declining balance.

Syntax

<OcxControl>.VDB(cost, salvage, life, start_period, end_period, factor, no_switch)

Cost is the initial cost of the asset.
Salvage is the value at the end of the depreciation (sometimes called the salvage value of the asset).
Life is the number of periods over which the asset is being depreciated (sometimes called the useful life of the asset).
Start_period is the starting period for which you want to calculate the depreciation. Start_period must use the same units as life.
End_period is the ending period for which you want to calculate the depreciation. End_period must use the same units as life.

Factor is the rate at which the balance declines. If factor is omitted, it is assumed to be 2 (the double-declining balance method). Change factor if you do not want to use the double-declining balance method. For a description of the double-declining balance method, see DDB.
No_switch is a logical value specifying whether to switch to straight-line depreciation when depreciation is greater than the declining balance calculation.

If no_switch is FALSE or omitted, Microsoft Excel switches to straight-line depreciation when depreciation is greater than the declining balance calculation.

All arguments except no_switch must be positive numbers.

Note : Functions are not case sensitive.

Examples

Suppose a factory purchases a new machine. The machine costs $2400 and has a lifetime of 10 years. The salvage value of the machine is $300. The following examples show depreciation over several periods. The results are rounded to two decimal places.
<OcxControl>.VDB(2400, 300, 3650, 0), 1) equals $1.32, the first day's depreciation.
<OcxControl>.VDB(2400, 300, 120, 0), 1) equals $40.00, the first month's depreciation.
<OcxControl>.VDB(2400, 300, 10, 0), 1) equals $480.00, the first year's depreciation.
<OcxControl>.VDB(2400, 300, 120, 6, 18) equals $396.31, the depreciation between the 6th month and the 18th month.
<OcxControl>.VDB(2400, 300, 120, 6, 18, 1.5) equals $311.81, the depreciation between the 6th month and the 18th month using a factor of 1.5 instead of the double-declining balance method.

PMT

Returns the periodic payment for an annuity based on constant payments and a constant interest rate.

Syntax

<OcxControl>.PMT(rate, nper, pv, fv, type)

For a more complete description of the arguments in PMT, see PV.
Rate is the interest rate per period.
Nper is the total number of payment periods in an annuity.
Pv is the present value—the total amount that a series of future payments is worth now.
Fv is the future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0).
Type is the number 0 or 1 and indicates when payments are due.

Set type equal to	If payments are due
0 or omitted	At the end of the period
1	At the beginning of the period

Note : Functions are not case sensitive.

Remarks
The payment returned by PMT includes principal and interest but no taxes, reserve payments, or fees sometimes associated with annuities.
Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 12%/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.

Tip To find the total amount paid over the duration of the annuity, multiply the returned PMT value by nper.

Examples

The following macro formula returns the monthly payment on a $10,000 loan at an annual rate of 8% that you must pay off in 10 months:
PMT(8%/12, 10, 10000) equals -$1037.03
For the same loan, if payments are due at the beginning of the period, the payment is:
PMT(8%/12, 10, 10000, 0), 1) equals -$1030.16
The following macro formula returns the amount someone must pay to you each month if you loan that person $5000 at 12% and want to be paid back in five months:
<OcxControl>.PMT(12%/12, 5, -5000) equals $1030.20.

PPMT

Returns the payment on the principal for a given period for an investment based on periodic, constant payments and a constant interest rate.

Syntax

<OcxControl>.PPMT(rate, per, nper, pv, fv, type)

For a more complete description of the arguments in PPMT, see PV.
Rate is the interest rate per period.
Per specifies the period and must be in the range 1 to nper.
Nper is the total number of payment periods in an annuity.
Pv is the present value—the total amount that a series of future payments is worth now.
Fv is the future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0).

Type is the number 0 or 1 and indicates when payments are due.

Set type equal to	If payments are due
0 or omitted	At the end of the period
1	At the beginning of the period

Note : Functions are not case sensitive.

Remarks

Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 12%/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.

Examples

The following formula returns the principal payment for the first month of a two-year $2000 loan at 10% annual interest:
<OcxControl>.PPMT(10%/12, 1, 24, 2000) equals -$75.62.

IPMT

Returns the interest payment for a given period for an investment based on periodic, constant payments and a constant interest rate. For a more complete description of the arguments in IPMT and for more information on annuity functions, see PV.

Syntax

<OcxControl>.IPMT(rate, per, nper, pv, fv, type)

Rate is the interest rate per period.
Per is the period for which you want to find the interest, and must be in the range 1 to nper.
Nper is the total number of payment periods in an annuity.
Pv is the present value, or the lump-sum amount that a series of future payments is worth right now.
Fv is the future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0).

Type is the number 0 or 1 and indicates when payments are due. If type is omitted, it is assumed to be 0.

Set type equal to	If payments are due
0	At the end of the period
1	At the beginning of the period

Note : Functions are not case sensitive.

Remarks
Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 12%/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.
For all the arguments, cash you pay out, such as deposits to savings, is represented by negative numbers; cash you receive, such as dividend checks, is represented by positive numbers.

Examples

The following formula calculates the interest due in the first month of a three-year $8000 loan at 10 percent annual interest:
<OcxControl>.IPMT(0.1/12, 1, 36, 8000) equals -$66.67

PV

Returns the present value of an investment. The present value is the total amount that a series of future payments is worth now. For example, when you borrow money, the loan amount is the present value to the lender.

Syntax

<OcxControl>.PV(rate, nper, pmt, fv, type)

Rate is the interest rate per period. For example, if you obtain an automobile loan at a 10% annual interest rate and make monthly payments, your interest rate per month is 10%/12, or 0.83%. You would enter 10%/12, or 0.83%, or 0.0083, into the formula as the rate.
Nper is the total number of payment periods in an annuity. For example, if you get a four-year car loan and make monthly payments, your loan has 4*12 (or 48) periods. You would enter 48 into the formula for nper.

Pmt is the payment made each period and cannot change over the life of the annuity. Typically, pmt includes principal and interest but no other fees or taxes. For example, the monthly payments on a $10,000, four-year car loan at 12% are $263.33. You would enter -263.33 into the formula as the pmt.
Fv is the future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0). For example, if you want to save $50,000 to pay for a special project in 18 years, then $50,000 is the future value. You could then make a conservative guess at an interest rate and determine how much you must save each month.

Type is the number 0 or 1 and indicates when payments are due.

Set type equal to	If payments are due
0 or omitted	At the end of the period
1	At the beginning of the period

Note : Functions are not case sensitive.

Remarks
Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12% annual interest, use 12%/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.
The following functions apply to annuities:

CUMIPMT	PPMT
CUMPRINC	PV
FV	RATE
FVSCHEDULE	XIRR
IPMT	XNPV
PMT	

An annuity is a series of constant cash payments made over a continuous period. For example, a car loan or a mortgage is an annuity. For more information, see the description for each annuity function.

In annuity functions, cash you pay out, such as a deposit to savings, is represented by a negative number; cash you receive, such as a dividend check, is represented by a positive number. For example, a $1000 deposit to the bank would be represented by the argument -1000 if you are the depositor and by the argument 1000 if you are the bank.
Microsoft Excel solves for one financial argument in terms of the others. If rate is not 0, then:

If rate is 0, then:
(pmt * nper) + pv + fv = 0

Example

Suppose you're thinking of buying an insurance annuity that pays $500 at the end of every month for the next 20 years. The cost of the annuity is $60,000 and the money paid out will earn 8%. You want to determine whether this would be a good investment. Using the PV function you find that the present value of the annuity is:
<OcxControl>.PV(0.08/12, 12*20, 500, , 0)) equals -$59,777.15.

FV

Returns the future value of an investment based on periodic, constant payments and a constant interest rate.

Syntax

<OcxControl>.FV(rate, nper, pmt, pv, type)

For a more complete description of the arguments in FV and for more information on annuity functions, see PV.
Rate is the interest rate per period.
Nper is the total number of payment periods in an annuity.
Pmt is the payment made each period; it cannot change over the life of the annuity. Typically, pmt contains principal and interest but no other fees or taxes.
Pv is the present value, or the lump-sum amount that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0.

Type is the number 0 or 1 and indicates when payments are due. If type is omitted, it is assumed to be 0.

Set type equal to	If payments are due
0	At the end of the period
1	At the beginning of the period

Note : Functions are not case sensitive.

Remarks
Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 12%/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.
For all the arguments, cash you pay out, such as deposits to savings, is represented by negative numbers; cash you receive, such as dividend checks, is represented by positive numbers.

Examples

<OcxControl>.FV(0.5%, 10, -200, -500, 1) equals $2581.40
<OcxControl>.FV(1%, 12, -1000) equals $12,682.50
<OcxControl>.FV(11%/12, 35, -2000, , 1) equals $82,846.25.

RATE

Returns the interest rate per period of an annuity. RATE is calculated by iteration and can have zero or more solutions. If the successive results of RATE do not converge to within 0.0000001 after 20 iterations, RATE returns the #NUM! error value.

Syntax

<OcxControl>.RATE(nper, pmt, pv, fv, type, guess)

See PV for a complete description of the arguments nper, pmt, pv, fv, and type.
Nper is the total number of payment periods in an annuity.
Pmt is the payment made each period and cannot change over the life of the annuity. Typically, pmt includes principal and interest but no other fees or taxes.
Pv is the present value—the total amount that a series of future payments is worth now.
Fv is the future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0).

Type is the number 0 or 1 and indicates when payments are due.

Set type equal to	If payments are due
0 or omitted	At the end of the period
1	At the beginning of the period

Guess is your guess for what the rate will be.
If you omit guess, it is assumed to be 10%.
If RATE does not converge, try different values for guess. RATE usually converges if guess is between 0 and 1.

Note : Functions are not case sensitive.

Remarks
Make sure that you are consistent about the units you use for specifying guess and nper. If you make monthly payments on a four-year loan at 12% annual interest, use 12%/12 for guess and 4*12 for nper. If you make annual payments on the same loan, use 12% for guess and 4 for nper.

Example

To calculate the rate of a four-year $8000 loan with monthly payments of $200:
<OcxControl>.RATE(48, -200, 8000) equals 0.77%.

NPER

Returns the number of periods for an investment based on periodic, constant payments and a constant interest rate.

Syntax

<OcxControl>.NPER(rate, pmt, pv, fv, type)

For a more complete description of the arguments in NPER and for more information about annuity functions, see PV.
Rate is the interest rate per period.
Pmt is the payment made each period; it cannot change over the life of the annuity. Typically, pmt contains principal and interest but no other fees or taxes.
Pv is the present value, or the lump-sum amount that a series of future payments is worth right now.
Fv is the future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0).

Type is the number 0 or 1 and indicates when payments are due.

Set type equal to	If payments are due
0 or omitted	At the end of the period
1	At the beginning of the period

Note : Functions are not case sensitive.

Examples

<OcxControl>.NPER(12%/12, -100, -1000, 10000, 1) equals 60
<OcxControl>.NPER(1%, -100, -1000, 10000) equals 60
<OcxControl>.NPER(1%, -100, 1000) equals 11.

SLN

Returns the straight-line depreciation of an asset for one period.

Syntax

<OcxControl>.SLN(cost, salvage, life)

Cost is the initial cost of the asset.
Salvage is the value at the end of the depreciation (sometimes called the salvage value of the asset).
Life is the number of periods over which the asset is being depreciated (sometimes called the useful life of the asset).

Note : Functions are not case sensitive.

Example

Suppose you've bought a truck for $30,000 that has a useful life of 10 years and a salvage value of $7500. The depreciation allowance for each year is:
<OcxControl>.SLN(30000, 7500, 10) equals $2250.

SYD

Returns the sum-of-years' digits depreciation of an asset for a specified period.

Syntax

<OcxControl>.SYD(cost, salvage, life, per)

Cost is the initial cost of the asset.
Salvage is the value at the end of the depreciation (sometimes called the salvage value of the asset).
Life is the number of periods over which the asset is being depreciated (sometimes called the useful life of the asset).
Per is the period and must use the same units as life.

Note : Functions are not case sensitive.

Remarks
SYD is calculated as follows:

Examples

If you've bought a truck for $30,000 that has a useful life of 10 years and a salvage value of $7500, the yearly depreciation allowance for the first year is:
<OcxControl>.SYD(30000,7500,10,1) equals $4090.91
The yearly depreciation allowance for the 10th year is:
<OcxControl>.SYD(30000,7500,10,10) equals $409.09.

AND

Returns TRUE if all its arguments are TRUE; returns FALSE if one or more arguments is FALSE.

Syntax

<OcxControl>.AND(logical1, logical2, ...)

Logical1, logical2,...	are 1 to 30 conditions you want to test that can be either TRUE or FALSE.

The arguments should be logical values or arrays or references that contain logical values.
If the specified range contains no logical values, AND returns the #VALUE! error value.

Note : Functions are not case sensitive.

Examples

<OcxControl>.AND(TRUE, TRUE) equals TRUE
<OcxControl>.AND(TRUE, FALSE) equals FALSE
<OcxControl>.AND(2+2=4, 2+3=5) equals TRUE.

FALSE

Returns the logical value FALSE.

Syntax

<OcxControl>.FALSE( )

Note : Functions are not case sensitive.

IF

Returns one value if logical_test evaluates to TRUE and another value if it evaluates to FALSE.

Syntax 1

Note : Functions are not case sensitive.

Remarks

You could use the following nested IF function:

```
<OcxControl>.IF(Average>89,"A",IF(Average>79,"B",
<OcxControl>.IF(Average>69,"C",IF(Average>59,"D","F"))))
```

In the preceding example, the second IF statement is also the value_if_false argument to the first IF statement. Similarly, the third IF statement is the value_if_false argument to the second IF statement. For example, if the first logical_test (Average>89) is TRUE, "A" is returned. If the first logical_test is FALSE, the second IF statement is evaluated, and so on.

NOT

Reverses the value of its argument. Use NOT when you want to make sure a value is not equal to one particular value.

Syntax

<OcxControl>.NOT(logical)

Logical is a value or expression that can be evaluated to TRUE or FALSE. If logical is FALSE, NOT returns TRUE; if logical is TRUE, NOT returns FALSE.

Note : Functions are not case sensitive.

Examples

<OcxControl>.NOT(FALSE) equals TRUE
<OcxControl>.NOT(1+1=2) equals FALSE.

OR

Returns TRUE if any argument is TRUE; returns FALSE if all arguments are FALSE.

Syntax

<OcxControl>.OR(logical1, logical2, ...)

Logical1, logical2,...	are 1 to 30 conditions you want to test that can be either TRUE or FALSE.

The arguments should be logical values or arrays or references that contain logical values.
If the specified range contains no logical values, OR returns the #VALUE! error value.
You can use an OR array formula to see if a value occurs in an array. To enter the OR formula as an array, press CTRL+SHIFT (in Microsoft Excel for Windows) or COMMAND+SHIFT

Note : Functions are not case sensitive.

Examples

<OcxControl>.OR(TRUE) equals TRUE
<OcxControl>.OR(1+1=1,2+2=5) equals FALSE.

TRUE

Returns the logical value TRUE.

Syntax

<OcxControl>.TRUE( )

Note : Functions are not case sensitive.

CHAR

Returns the character specified by the code number. Use CHAR to translate code numbers you might get from files on other types of computers into characters.

Syntax

<OcxControl>.CHAR(number)

Number is a number between 1 and 255 specifying which character you want. The character is from the character set used by your computer.

Operating environment	Character set
Macintosh	Macintosh character set
Windows	ANSI

Note : Functions are not case sensitive.

Examples

<OcxControl>.CHAR(65) equals "A"
<OcxControl>.CHAR(33) equals "!".

CLEAN

Removes all nonprintable characters from text. Use CLEAN on text imported from other applications which contains characters that may not print with your operating system. For example, you can use CLEAN to remove some low-level computer code that is frequently at the beginning and end of data files and cannot be printed.

Syntax

<OcxControl>.CLEAN(text)

Note : Functions are not case sensitive.

Example

Since CHAR(7) returns a nonprintable character:
<OcxControl>.CLEAN(CHAR(7)&"text"&CHAR(7)) equals "text".

CODE

Returns a numeric code for the first character in a text string. The returned code corresponds to the character set used by your computer.

Syntax

<OcxControl>.CODE(text)

Operating environment	Character set
Macintosh	Macintosh character set
Windows	ANSI

Text is the text for which you want the code of the first character.

Note : Functions are not case sensitive.

Examples

<OcxControl>.CODE("A") equals 65
<OcxControl>.CODE("Alphabet") equals 65.

CONCATENATE

Joins several text items into one text item.

Syntax

<OcxControl>.CONCATENATE (text1, text2, ...)
Text1, text2,...   are 1 to 30 text items to be joined into a single text item.   The text items can be text strings, numbers, or single-cell references.

Note : Functions are not case sensitive.

Remarks
The "&" operator can be used instead of CONCATENATE to join text items.

Examples

<OcxControl>.CONCATENATE("Total ", "Value") equals "Total Value".   This is equivalent to typing "Total"&" "&"Value" .

DOLLAR

Converts a number to text using currency format, with the decimals rounded to the specified place. The format used is $#,##0.00_);($#,##0.00).

Syntax

<OcxControl>.DOLLAR(number, decimals)

Number is a number, a reference to a cell containing a number, or a formula that evaluates to a number.
Decimals is the number of digits to the right of the decimal point. If decimals is negative, number is rounded to the left of the decimal point. If you omit decimals, it is assumed to be 2.

Note : Functions are not case sensitive.

Examples

<OcxControl>.DOLLAR(1234.567, 2) equals "$1234.57"
<OcxControl>.DOLLAR(1234.567, -2) equals "$1200"
<OcxControl>.DOLLAR(-1234.567, -2) equals "($1200)"
<OcxControl>.DOLLAR(-0.123, 4) equals "($0.1230)"
<OcxControl>.DOLLAR(99.888) equals "$99.89".

EXACT

Compares two text strings and returns TRUE if they are exactly the same, FALSE otherwise. EXACT is case-sensitive but ignores formatting differences. Use EXACT to test text being entered onto a document.

Syntax

<OcxControl>.EXACT(text1, text2)

Text1	is the first text string.
Text2	is the second text string.

Note : Functions are not case sensitive.

Examples

<OcxControl>.EXACT("word", "word") equals TRUE
<OcxControl>.EXACT("Word", "word") equals FALSE
<OcxControl>.EXACT("w ord", "word") equals FALSE

FIND

Finds one string of text within another string of text and returns the number of the character at which find_text first occurs. You can also use SEARCH to find one string of text within another, but unlike SEARCH, FIND is case-sensitive and doesn't allow wildcard characters.

Syntax

<OcxControl>.FIND(find_text, within_text, start_num)

Find_text is the text you want to find.

If find_text is "" (empty text), FIND matches the first character in the search string (that is, the character numbered start_num or 1).
Find_text cannot contain any wildcard characters.

Within_text is the text containing the text you want to find.
Start_num specifies the character at which to start the search. The first character in within_text is character number 1. If you omit start_num, it is assumed to be 1.

Note : Functions are not case sensitive.

Remarks
If find_text does not appear in within_text, FIND returns the #VALUE! error value.
If start_num is not greater than zero, FIND returns the #VALUE! error value.
If start_num is greater than the length of within_text, FIND returns the #VALUE! error value.

Examples

<OcxControl>.FIND("M","Miriam McGovern") equals 1
<OcxControl>.FIND("m","Miriam McGovern") equals 6
<OcxControl>.FIND("M","Miriam McGovern",3) equals 8.

LEFT

Returns the first (or leftmost) character or characters in a text string.

Syntax

<OcxControl>.LEFT(text, num_chars)

Text	is the text string containing the characters you want to extract.
Num_chars	specifies how many characters you want LEFT to return.

Num_chars must be greater than or equal to zero.
If num_chars is greater than the length of text, LEFT returns all of text.
If num_chars is omitted, it is assumed to be 1.

Note : Functions are not case sensitive.

Examples

<OcxControl>.LEFT("Sale Price", 4) equals "Sale".

LEN

Returns the number of characters in a text string.

Syntax

<OcxControl>.LEN(text)

Text is the text whose length you want to find. Spaces count as characters.

Note : Functions are not case sensitive.

Examples

<OcxControl>.LEN("Phoenix, AZ") equals 11
<OcxControl>.LEN("") equals 0.

LOWER

Converts all uppercase letters in a text string to lowercase.

Syntax

<OcxControl>.LOWER(text)

Text is the text you want to convert to lowercase. LOWER does not change characters in text that are not letters.

Note : Functions are not case sensitive.

Examples

<OcxControl>.LOWER("E. E. Cummings") equals "e. e. cummings"
<OcxControl>.LOWER("Apt. 2B") equals "apt. 2b"
<OcxControl>.LOWER is similar to PROPER and UPPER. Also see examples for PROPER.

MID

Returns a specific number of characters from a text string, starting at the position you specify.

Syntax

<OcxControl>.MID(text, start_num, num_chars)

Text is the text string containing the characters you want to extract.
Start_num is the position of the first character you want to extract in text. The first character in text has start_num 1, and so on.

If start_num is greater than the length of text, MID returns "" (empty text).
If start_num is less than the length of text, but start_num plus num_chars exceeds the length of text, MID returns the characters up to the end of text.
If start_num is less than 1, MID returns the #VALUE! error value.

Num_chars specifies how many characters to return from text. If num_chars is negative, MID returns the #VALUE! error value.

Note : Functions are not case sensitive.

Examples

<OcxControl>.MID("Fluid Flow", 1, 5) equals "Fluid"
<OcxControl>.MID("Fluid Flow", 7, 20) equals "Flow"
<OcxControl>.MID("1234", 5, 5) equals "" (empty text).

PROPER

Capitalizes the first letter in text and any other letters in text that follow any character other than a letter. Converts all other letters to lowercase.

Syntax

<OcxControl>.PROPER(text)

Note : Functions are not case sensitive.

Examples

<OcxControl>.PROPER("this is a TITLE") equals "This Is A Title"
<OcxControl>.PROPER("2-cent's worth") equals "2-Cent'S Worth"
<OcxControl>.PROPER("76BudGet") equals "76Budget".

REPLACE

Replaces part of a text string with a different text string.

Syntax

<OcxControl>.REPLACE(old_text, start_num, num_chars, new_text)

Old_text	is text in which you want to replace some characters.
Start_num	is the position of the character in old_text that you want to replace with new_text.
Num_chars	is the number of characters in old_text that you want to replace with new_text.
New_text	is the text that will replace characters in old_text.

<u>Note :</u> Functions are not case sensitive.

Examples

The following formula replaces five characters with new_text, starting with the sixth character in old_text:
<OcxControl>.REPLACE("abcdefghijk", 6, 5, "*") equals "abcde*k"
The sixth through tenth characters are all replaced by "*".
The following formula replaces the last two digits of 1990 with 91:
<OcxControl>.REPLACE("1990", 3, 2, "91") equals "1991".

REPT

Repeats text a given number of times the instances of a text string.

Syntax

<OcxControl>.REPT(text, number_times)

Note : Functions are not case sensitive.

Text is the text you want to repeat.
Number_times is a positive number specifying the number of times to repeat text. If number_times is 0, REPT returns "" (empty text). If number_times is not an integer, it is truncated. The result of the REPT function cannot be longer than 255 characters.

Tip You can use this function to create a simple histogram on your worksheet.

Examples

<OcxControl>.REPT("*-", 3) equals "*-*-*-".

RIGHT

Returns the last (or rightmost) character or characters in a text string.

Syntax

<OcxControl>.RIGHT(text, num_chars)

Text	is the text string containing the characters you want to extract.
Num_chars	specifies how many characters you want to extract.

Num_chars must be greater than or equal to zero.

If num_chars is greater than the length of text, RIGHT returns all of text.

If num_chars is omitted, it is assumed to be 1.

Note : Functions are not case sensitive.

Examples

<OcxControl>.RIGHT("Sale Price", 5) equals "Price"
<OcxControl>.RIGHT("Stock Number") equals "r".

SEARCH

Returns the number of the character at which a specific character or text string is first found, reading from left to right. Use SEARCH to discover the location of a character or text string within another text string, so that you can use the MID or REPLACE functions to change the text.

Syntax

<OcxControl>.SEARCH(find_text, within_text, start_num)

Find_text is the text you want to find. You can use the wildcard characters, question mark (?) and asterisk (*), in find_text. A question mark matches any single character; an asterisk matches any sequence of characters. If you want to find an actual question mark or asterisk, type a tilde (~) before the character. If find_text is not found, the #VALUE! error value is returned.
Within_text is the text in which you want to search for find_text.
Start_num is the character number in within_text, counting from the left, at which you want to start searching.

If start_num is omitted, it is assumed to be 1.
If start_num is not greater than 0 or is greater than the length of within_text, the #VALUE! error value is returned.

Tip Use start_num to skip a specified number of characters from the left of the text. For example, suppose you are working with a text string such as "AYF0093.YoungMensApparel". To find the number of the first "Y" in the descriptive part of the text string, set start_num equal to 8 so that the serial-number portion of the text is not searched. SEARCH begins with character 8, finds find_text at the next character, and returns the number 9. SEARCH always returns the number of characters from the left of the text string, not from start_num.

Note : Functions are not case sensitive.

Remarks
SEARCH does not distinguish between uppercase and lowercase letters when searching text.
SEARCH is similar to FIND, except that FIND is case-sensitive.

Examples

<OcxControl>.SEARCH("e","Statements",6) equals 7.

SUBSTITUTE

Substitutes new_text for old_text in a text string. Use SUBSTITUTE when you want to replace specific text in a text string; use REPLACE when you want to replace any text that occurs in a specific location in a text string.

Syntax

<OcxControl>.SUBSTITUTE(text, old_text, new_text, instance_num)

Text is the text or the reference to a cell containing text for which you want to substitute characters.
Old_text is the text you want to replace.
New_text is the text you want to replace old_text with.
Instance_num specifies which occurrence of old_text you want to replace with new_text. If you specify instance_num, only that instance of old_text is replaced. Otherwise, every occurrence of old_text in text is changed to new_text.

Note : Functions are not case sensitive.

Examples

<OcxControl>.SUBSTITUTE("Sales Data", "Sales", "Cost") equals "Cost Data"
<OcxControl>.SUBSTITUTE("Quarter 1, 1991", "1", "2", 1) equals "Quarter 2, 1991"
<OcxControl>.SUBSTITUTE("Quarter 1, 1991", "1", "2", 3) equals "Quarter 1, 1992".

TRIM

Removes all spaces from text except for single spaces between words. Use TRIM on text that you have received from another application that may have irregular spacing.

Syntax

<OcxControl>.TRIM(text)

Text   is the text from which you want spaces removed.

Note : Functions are not case sensitive.

Example

<OcxControl>.TRIM(" First   Quarter   Earnings   ") equals "First Quarter Earnings".

UPPER

Converts text to uppercase.

Syntax

<OcxControl>.UPPER(text)

Text is the text you want converted to uppercase. Text can be a reference or text string.

Note : Functions are not case sensitive.

Examples

<OcxControl>.UPPER("total") equals "TOTAL".

FIX

Rounds a number downwards using absolute value top the nearest integer.

Syntax

<OcxControl>.FIX(number)

Number   is the number you want to round downwards.

Note : Functions are not case sensitive.

Examples

<OcxControl>.FIX(1.1) = 1
<OcxControl>.FIX(-1.1) = -1.

COMPLEXADD

Adds two complex numbers together.They are expressed in terms of their real and imaginary parts: a + ib where a is the square root of -1, a and b are both
real numbers.

Syntax

<OcxControl>.ComplexAdd(a1,b1,a2,b2,a3,b3)

Note : Functions are not case sensitive.

Example

For two complex numbers: (2 + 3i) + (4 + 7i)
<OcxControl>.ComplexAdd(2,3,4,7,A,B);
A = 6
B = 10.

COMPLEXMULT

Multiplies two complex numbers together.They are expressed in terms of their real and imaginary parts: a + ib where a is the square root of -1, a and b are both
real numbers.

Syntax

<OcxControl>.ComplexMult(a1,b1,a2,b2,a3,b3)

The function works like this:

a3 + ib3 = (a1 + ib1) * (a2 - ib2) = ((a1*a2) + (b1*b2)) - ((a2 * b1) - (a1 * b2))**I**
a3 = ((a1*a2) + (b1*b2))
b3 = ((a2 * b1) - (a1 * b2))**I**

Note : Functions are not case sensitive.

Example

For two complex numbers: (2 + 3i) + (4 + 7i)
<OcxControl>.ComplexMult(2,3,4,7,A,B);
A = 29
B = -2.

COMPLEXSU

Adds two complex numbers together.They are expressed in terms of their real and imaginary parts: a + ib where a is the square root of -1, a and b are both
real numbers.

Syntax

<OcxControl>.ComplexAdd(a1,b1,a2,b2,a3,b3)

The function works like this:

a3 = (a1 - a2)
b3 = (b1 - b2)

Note : Functions are not case sensitive.

Example

For two complex numbers: (2 + 3i) + (4 + 7i)
<OcxControl>.ComplexAdd(2,3,4,7,A,B);
A = -2
B = -4.

BITS LEFT

Takes a Long Integer and treats it as a string of 32 bits. It shifts the bits to the left.

Syntax

Long Integer =<OcxControl>. BitsLeft(integer,integer)

Note : Functions are not case sensitive.

Remarks
Takes a long integer and shifts the bits n bits to the left.

Example

if a variable I = 00000000 00000000 00000000 00001000 = 8 decimal, then
<OcxControl>.BitsLeft(I,2) = 00000000 00000000 00000000 00100000 = 32 decimal.

BITS RIGHT

Takes a Long Integer and treats it as a string of 32 bits. It shifts the bits to the right.

Syntax

Long Integer =<OcxControl>. BitsRight (integer,integer)

Note : Functions are not case sensitive.

Remarks
Takes a long integer and shifts the bits n bits to the right.

Example

if a variable I = 00000000 00000000 00000000 00001000 = 8 decimal, then
<OcxControl>.BitsRight (I,2) = 00000000 00000000 00000000 00000010 = 2 decimal.

BITS OFF

Turns an individual bit off.

Syntax

Long = <OcxControl>.BitsOff(Long,position)

Note : Functions are not case sensitive.

Remarks
Takes a long integer and treats it like a string of 32 bits.Bitsoff turns the bit at the specified position to zero.

Example

if a variable **l** = 00000000 00000000 00000000 11111111 then
<OcxControl>.BitsOff(**l**,4) = 00000000 00000000 00000000 11110111.

BITS ON

Turns an individual bit on.

Syntax

<OcxControl>.Long =   BitsOn(Long,position)

<u>Note : </u>Functions are not case sensitive.

Remarks
Takes a long integer and treats it like a string of 32 bits.Bitsoff turns the bit at the specified position to one.

Example

if a variable **l** = 00000000 00000000 00000000 11111111   then
<OcxControl>.BitsOn(**l**,9) = 00000000 00000000 00000001 11110111.

AVEDEV

Returns the average of the absolute deviations of data points from their mean. AVEDEV is a measure of the variability in a data set.

Syntax

<OcxControl>.Avedev (number1, number2, ...)

Number1, number2,...   are 1 to 30 arguments for which you want the average of the absolute deviations. You can also use a single array or a reference to an array instead of arguments separated by commas.

Note : Functions are not case sensitive.

Remarks

The arguments should be numbers, or names, arrays, or references that contain numbers.
The equation for average deviation is:

AVEDEV is influenced by the unit of measurement in the input data.

Example

<OcxControl>.Avedev(4, 5, 6, 7, 5, 4, 3) equals 1.020408.

AVERAGE

Returns the average (arithmetic mean) of the arguments.

Syntax

<OcxControl>.Average(number1, number2, ...)

Number1, number2,...	are 1 to 30 numeric arguments for which you want the average.

Note : Functions are not case sensitive.

Remarks

The arguments should be numbers, or names, arrays, or references that contain numbers.

Examples

<OcxControl>.Average(4,5,6,11) equals 26

KURT

Returns the kurtosis of a data set. Kurtosis characterizes the relative peakedness or flatness of a distribution compared to the normal distribution. Positive kurtosis indicates a relatively peaked distribution. Negative kurtosis indicates a relatively flat distribution.

Syntax

<OcxControl>.Kurt(number1, number2, ...)

Number1,number2,... are 1 to 30 arguments for which you want to calculate kurtosis. You can also use a single array or a reference to an array instead of arguments separated by commas.

<u>Note :</u> Functions are not case sensitive.

Remarks

The arguments should be numbers, or names, arrays, or references that contain numbers.
If there are less than four data points, or if the standard deviation of the sample equals zero, KURT returns the #DIV/0! error value.
Kurtosis is defined as:

where:
s is the sample standard deviation.

Example

<OcxControl>.Kurt (3,4,5,2,3,4,5,6,4,7) returns -0.1518

MAX

Returns the maximum value in a list of arguments.

Syntax

<OcxControl>.Max(number1, number2, ...)

Number1, number2,...	are 1 to 30 numbers for which you want to find the maximum value.

Note : Functions are not case sensitive.

Examples

If <OcxControl>.Max(10, 7, 9, 27, 2) gives 27.

MEDIAN

Returns the median of the given numbers. The median is the number in the middle of a set of numbers; that is, half the numbers have values that are greater than the median and half have values that are less.

Syntax

<OcxControl>.Median(number1, number2, ...)

Number1, number2,...	are 1 to 30 numbers for which you want the median.

The arguments should be numbers or names, arrays, or references that contain numbers.

Note : Functions are not case sensitive.

Remarks

If there is an even number of numbers in the set, then MEDIAN calculates the average of the two numbers in the middle. See the second example following.

Examples

<OcxControl>.Median(1, 2, 3, 4, 5) equals 3
<OcxControl>.Median(1, 2, 3, 4, 5, 6) equals 3.5, the average of 3 and 4.

MIN

Returns the smallest number in the list of arguments.

Syntax

<OcxControl>.Min(number1, number2, ...)

Number1, number2,...   are 1 to 30 numbers for which you want to find the minimum value.

If the arguments contain no numbers, MIN returns 0.

Note : Functions are not case sensitive.

Examples

If <OcxControl>.Min(10, 7, 9, 27, 2) equals 2.

MODE

Returns the most frequently occurring value in an array or range of data. Like MEDIAN, MODE is a location measure

Syntax

<OcxControl>.Mode(number1, number2, ...)

Number1, number2,... are 1 to 30 arguments for which you want to calculate the mode. You can also use a single array or a reference to an array instead of arguments separated by commas.

Note : Functions are not case sensitive.

Remarks

The arguments should be numbers, or names, arrays, or references that contain numbers.
If the data set contains no duplicate data points, MODE returns the #N/A error value.

The mode is the most frequently occurring value; the median is the middle value; and the mean is the average value. No single measure of central tendency provides a complete picture of the data. Suppose data is clustered in three areas, half around a single low value, and half around two large values. Both AVERAGE and MEDIAN may return a value in the relatively empty middle, while MODE may return the dominant low value.

Example

<OcxControl>.Mode ({5.6, 4, 4, 3, 2, 4}) equals 4.

PERMUT

Returns the number of permutations for a given number of objects that can be selected from number objects. A permutation is any set or subset of objects or events where internal order is significant. Permutations are different than combinations, for which the internal order is not significant. Use this function for lottery-style probability calculations.

Syntax

<OcxControl>.Permut (number, number_chosen)

Number	is an integer that describes the number of objects.
Number_chosen	is an integer that describes the number of objects in each permutation.

Note : Functions are not case sensitive.

Remarks

Both arguments are truncated to integers.

If number or number_chosen is non-numeric, PERMUT returns the #VALUE! error value.
If number £ 0 or if number_chosen < 0, PERMUT returns the #NUM! error value.
If number < number_chosen, PERMUT returns the #NUM! error value.
The equation for the number of permutations is:

Example

Suppose you want to calculate the odds of selecting a winning lottery number. Each lottery number contains three numbers, each of which can be between 0 and 99, inclusive. The following function calculates the number of possible permutations.

<OcxControl>.Permut (100,3) equals 970,200.

POISSON

Returns the Poisson distribution. A common application of the Poisson distribution is predicting the number of events over a specific time, such as the number of cars arriving at a toll plaza in one minute.

Syntax

<OcxControl>.Poisson(x, mean, cumulative)

X is the number of events.
Mean is the expected numeric value.
Cumulative is a logical value that determines the form of the probability distribution returned. If cumulative is TRUE, POISSON returns the cumulative Poisson probability that the number of random events occurring will be between zero and x inclusive; if FALSE, it returns the Poisson probability mass function that the number of events occurring will be exactly x.

Note : Functions are not case sensitive.

Remarks

If x is not an integer, it is truncated.
If x or mean is non-numeric, POISSON returns the #VALUE! error value.
If x £ 0, POISSON returns the #NUM! error value.
If mean £ 0, POISSON returns the #NUM! error value.
POISSON is calculated as follows.

For cumulative = FALSE:

For cumulative =TRUE:

Examples

<OcxControl>.Poisson (2,5,FALSE) equals 0.084224
<OcxControl>.Poisson(2,5,TRUE) equals 0.124652.

PROB

Returns the probability that values in a range are between two limits. If upper_limit is not supplied, returns the probability that values in x_range are equal to lower_limit.

Syntax

<OcxControl>.Prob(x_range, prob_range, lower_limit, upper_limit)

X_range	is the range of numeric values of x with which there are associated probabilities.
Prob_range	is a set of probabilities associated with values in x_range.
Lower_limit	is the lower bound on the value for which you want a probability.
Upper_limit	is the optional upper bound on the value for which you want a probability.

Note : Functions are not case sensitive.

Remarks

If any value in prob_range £ 0 or if any value in prob_range > 1, PROB returns the #NUM! error value.

If the sum of the values in prob_range ¹ 1, PROB returns the #NUM! error value.

If upper_limit is omitted, PROB returns the probability of being equal to lower_limit.

If x_range and prob_range contain a different number of data points, PROB returns the #N/A error value.

Examples

<OcxControl>.Prob ({0,1,2,3},{0.2,0.3,0.1,0.4},2) equals 0.1
<OcxControl>.Prob({0,1,2,3},{0.2,0.3,0.1,0.4},1,3) equals 0.8.

STDEV

Estimates standard deviation based on a sample. The standard deviation is a measure of how widely values are dispersed from the average value (the mean).

Syntax

<OcxControl>.Stdev(number1,number2,...)
Number1,number2,... are 1 to 30 number arguments corresponding to a sample of a population. You can also use a single array or a reference to an aray instead of arguments separated by commas.

Note : Functions are not case sensitive.

Remarks

STDEV assumes that its arguments are a sample of the population. If your data represents the entire population, you should compute the standard deviation using STDEVP.

The standard deviation is calculated using the "nonbiased" or "n-1" method.

STDEV uses the following formula:

Example

<OcxControl>.Stdev (1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) equals 27.46

STDEVP

Calculates standard deviation based on the entire population given as arguments. The standard deviation is a measure of how widely values are dispersed from the average value (the mean).

Syntax

<OcxControl>.Stdevp (number1,number2],...
Number1,number2,...   are 1 to 30 number arguments corresponding to a population. You can also use a single array or a reference to an aray instead of arguments separated by commas.

Note : Functions are not case sensitive.

Remarks

STDEVP assumes that its arguments are the entire population. If your data represents a sample of the population, you should compute the standard deviation using STDEV.

For large sample sizes, STDEV and STDEVP return approximately equal values.

The standard deviation is calculated using the "biased" or "n" method.

STDEVP uses the following formula:

Example

<OcxControl>.Stdevp(1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) equals 26.05

VAR

Estimates variance based on a sample.

Syntax

<OcxControl>.Var(number1, number2, ...)

Number1,number2,... are 1 to 30 number arguments corresponding to a sample of a population.

Note : Functions are not case sensitive.

Remarks

VAR assumes that its arguments are a sample of the population. If your data represents the entire population, you should compute the variance using VARP.
VAR uses the following formula:

Example

<OcxControl>.Var(1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) equals 754.3.

VARP

Calculates variance based on the entire population.

Syntax

<OcxControl>.Varp(number1, number2, ...)

Number1, number2,...	are 1 to 30 number arguments corresponding to a population.

Note : Functions are not case sensitive.

Remarks

VARP assumes that its arguments are the entire population. If your data represents a sample of the population, you should compute the variance using VAR.
The equation for VARP is :

Example

<OcxControl>.Varp (1345, 1301, 1368, 1322, 1310, 1370, 1318, 1350, 1303, 1299) equals 678.8.

NPV

Description

Returns the net present value of an investment based on a series of periodic cash flows and a discount rate. The net present value of an investment is today's value of a series of future payments (negative values) and income (positive values).

Usage
NPV(rate, value1, value2, ...)

Rate is the rate of discount over the length of one period.
Value1, value2,... are 1 to 29 arguments representing the payments and income.

Value1, value2,... must be equally spaced in time and occur at the end of each period.
NPV uses the order of value1, value2,... to interpret the order of cash flows. Be sure to enter your payment and income values in the correct sequence.
Examples
Suppose you're considering an investment in which you pay $10,000 one year from today and receive an annual income of $3000, $4200, and $6800 in the three years that follow. Assuming an annual discount rate of 10 percent, the net present value of this investment is:

NPV(10%, -10000, 3000, 4200, 6800) equals $1188.44

Example

See Also

FV, PV, RATE , NPER

NPV

Returns the net present value of an investment based on a series of periodic cash flows and a discount rate. The net present value of an investment is today's value of a series of future payments (negative values) and income (positive values).

Syntax
<OcxControl>.NPV(rate, value1, value2, ...)

Rate	is the rate of discount over the length of one period.
Value1, value2,...	are 1 to 29 arguments representing the payments and income.

Value1, value2,... must be equally spaced in time and occur at the end of each period.
NPV uses the order of value1, value2,... to interpret the order of cash flows. Be sure to enter your payment and income values in the correct sequence.

Note : Functions are not case sensitive.

Examples
Suppose you're considering an investment in which you pay $10,000 one year from today and receive an annual income of $3000, $4200, and $6800 in the three years that follow. Assuming an annual discount rate of 10 percent, the net present value of this investment is:

<OcxControl>.NPV(10%, -10000, 3000, 4200, 6800) equals $1188.44

IRR

Description

Returns the internal rate of return for a series of cash flows represented by the numbers in values. These cash flows do not have to be even, as they would be for an annuity. The internal rate of return is the interest rate received for an investment consisting of payments (negative values) and income (positive values) that occur at regular periods.

Usage
IRR(values, guess)

Values is an array you want to calculate the internal rate of return.

Values must contain at least one positive value and one negative value to calculate the internal rate of return.
IRR uses the order of values to interpret the order of cash flows. Be sure to enter your payment and income values in the sequence you want.

Examples

Suppose you want to start a restaurant business. You estimate it will cost $70,000 to start the business and expect to net the following income in the first five years: $12,000, $15,000, $18,000, $21,000, and $26,000. following values: $-70,000, $12,000, $15,000, $18,000, $21,000 and $26,000, respectively.
To calculate the investment's internal rate of return after four years:

IRR(12,000,18,000, 21,000,26,000) equals -2.12%

Example

See Also

MIRR

MIRR

Description

Returns the modified internal rate of return for a series of periodic cash flows. MIRR considers both the cost of the investment and the interest received on reinvestment of cash.

Usage
MIRR(values, finance_rate, reinvest_rate)

Values is an array or a reference to cells that contain numbers. These numbers represent a series of payments (negative values) and income (positive values) occurring at regular periods.

Values must contain at least one positive value and one negative value to calculate the modified internal rate of return. Otherwise, MIRR returns the #DIV/0! error value.
Finance_rate is the interest rate you pay on the money used in the cash flows.
Reinvest_rate is the interest rate you receive on the cash flows as you reinvest them.

Examples

Suppose you're a commercial fisherman just completing your fifth year of operation. Five years ago, you borrowed $120,000 at 10 percent annual interest to purchase a boat. Your catches have yielded $39,000, $30,000, $21,000, $37,000, and $46,000. During these years you reinvested your profits, earning 12% annually. In a worksheet, your loan amount is entered as -$120,000 in B1, and your five annual profits are entered in B2:B6.
To calculate the investment's modified rate of return after five years:

MIRR(39,000, 30,000, 21,000, 37,000, 46,000 10%, 12%) equals 12.61%

Example

See Also

IRR

IRR

Returns the internal rate of return for a series of cash flows represented by the numbers in values. These cash flows do not have to be even, as they would be for an annuity. The internal rate of return is the interest rate received for an investment consisting of payments (negative values) and income (positive values) that occur at regular periods.

Syntax

<OcxControl>.IRR(values, guess)

Values is an array you want to calculate the internal rate of return.

Values must contain at least one positive value and one negative value to calculate the internal rate of return.
IRR uses the order of values to interpret the order of cash flows. Be sure to enter your payment and income values in the sequence you want.

Note : Functions are not case sensitive.

Examples

Suppose you want to start a restaurant business. You estimate it will cost $70,000 to start the business and expect to net the following income in the first five years: $12,000, $15,000, $18,000, $21,000, and $26,000. following values: $-70,000, $12,000, $15,000, $18,000, $21,000 and $26,000, respectively.
To calculate the investment's internal rate of return after four years:

IRR(12,000,18,000, 21,000,26,000) equals -2.12%

AVERAGE

Returns the average of a series of numbers.

Usage
number = AVERAGE(number1, number2….)

Description
Takes a series of numbers and returns the average.

Example
AVERAGE(1,2,3,4,5) = 3

Note:
When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available

variant **= AVERAGEVB(*SafeArray*)**
In Visual Basic, the default array structure is of type SafeArray, and nothing special need be done. In other languages, you may need to set a variant variable equal to an array, or explicitly create a Safe Array datatype. Because different development environments handle arrays and variant data types differently, this function may not work properly in those environments.

double **= AVERAGE(*array of double, integer*)**
In most languages you should use this syntax. It takes a standard array of double and an integer variable which is the size of the array.

Examples

AVERAGE(10, 7, 9, 27, 2,) equals 11

Example

KURT

Returns the Curtosis of a set of numbers.

Usage

double = KURT(number1, number2….)

Description

The Curtosis of a set of numbers is a number that characterises it's relative sharpness or flatness based on a normal distribution. Positive values indicate a sharper peak to the curve, and a negative number indicates a flatter shape.

It must take a minimum of 4 numbers.

Example

KURT(1,2,3,4,5) = -1.2

Note:

When calling the function via the DLL or Func-O-Matic, please be aware of the variations of the function that are available:

***variant* = KURT(*safe array*)**

In Visual Basic, the default array structure is of type SafeArray, and nothing special need be done. In other languages, you may need to set a variant variable equal to an array, or explicitly create a Safe Array datatype. Because different development environments handle arrays and variant data types differently, this function may not work properly in those environments.

***double* = KURT(*array of double, size*)**

For other languages, you should use this syntax. It takes an array of double and an integer variable which is the size of the array

Example

Kurt (3,4,5,2,3,4,5,6,4,7) returns -0.1518

Example

AVERAGE

Returns the average (arithmetic mean) of the arguments.

Syntax

<OcxControl>.Average(number1, number2, ...)

Number1, number2,...   are 1 to 30 numeric arguments for which you want the average.

Note : Functions are not case sensitive.

Remarks

The arguments should be numbers, or names, arrays, or references that contain numbers.

Examples

<OcxControl>.Average(4,5,6,11) equals 26

KURT

Returns the kurtosis of a data set. Kurtosis characterizes the relative peakedness or flatness of a distribution compared to the normal distribution. Positive kurtosis indicates a relatively peaked distribution. Negative kurtosis indicates a relatively flat distribution.

Syntax

<OcxControl>.Kurt(number1, number2, ...)

Number1,number2,... are 1 to 30 arguments for which you want to calculate kurtosis. You can also use a single array or a reference to an array instead of arguments separated by commas.

<u>Note :</u> Functions are not case sensitive.

Remarks

The arguments should be numbers, or names, arrays, or references that contain numbers.

If there are less than four data points, or if the standard deviation of the sample equals zero, KURT returns the #DIV/0! error value.

Kurtosis is defined as:

where:
s is the sample standard deviation.

Example

<OcxControl>.Kurt (3,4,5,2,3,4,5,6,4,7) returns -0.1518

CREDITCARDTYPE

Takes a string representing a credit card number, and returns the type of card.

Usage

Integer = CreditCardType(String)

Description

There is an algorithm that can check to see if a credit card number is potentially valid, or just a series of random digits. Each type of credit card is given a certain series of potentially valid credit card numbers.

While this routine does not guarantee that the credit card is indeed valid, it does tell if the number is at least *potentially* valid. Only contacting a credit agency can determine if a given card has been issued. This routine will rule out approximately 9 out of 10 properly structured but randomly selected credit card numbers.

Spaces in the string are automatically stripped out for analysis.

Return Results are:
1=American Express
2=MasterCard
3=Visa
4=Other potentially valid number
5=invalid

Example

CreditCardType(“5431 1234 5678 9012”) = 5

Example

CREDITCARDTYPE

Takes a string representing a credit card number, and returns the type of card.

Syntax

<OcxControl>.CreditCardType(String)

Note : Functions are not case sensitive.

Description

While this routine does not guarantee that the credit card is indeed valid, it does tell if the number is at least *potentially* valid. Only contacting a credit agency can determine if a given card has been issued. This routine will rule out approximately 9 out of 10 properly structured but randomly selected credit card numbers.

Spaces in the string are automatically stripped out for analysis.

Return Results are:
1=American Express
2=MasterCard
3=Visa
4=Other potentially valid number
5=invalid

Example

<OcxControl>.CreditCardType("5431 1234 5678 9012") = 5

NUMBERTOWORDS

Description

Takes a number and translates it to a string in word format.

Usage

NumberToWords(number)

 where number is the argument to be converted to words.

Examples

NumToWords(134) = “One Hundred Thirty Four”

Example

NUMBERTOWORDS

Takes a number and translates it to a string in word format.

Syntax

<OcxControl>.NumberToWords(number)

where number is the argument to be converted to words.
Note : Functions are not case sensitive.

Examples

<OcxControl>.NumToWords(134) = “One Hundred Thirty Four”

ARABIC

Description

Takes a string given in roman and returns an integer.

Usage
Arabic(string)

where string is the argument to be converted to number.

Examples

Arabic(“XV”) =   15.

Example

See Also

ROMAN

ARABIC

Takes a string given in roman and returns an integer.

Usage
<OcxControl>.Arabic(string)

where string is the argument to be converted to number.
Note : Functions are not case sensitive.The maximum number can be upto 3999.

Examples

<OcxControl>.Arabic(“XV”) = 15.

SQR

Description

Takes a number and gives the square of the number returns an integer.

Usage
Sqr(number)

where number is the argument to be give the square of the number.

Examples

Sqr(5) =   25.

Example

See Also

SQRT

SQR

Description

Takes a number and gives the square of the number returns an integer.

Usage
<OcxControl>.Sqr(number)

where number is the argument to be give the square of the number.
<u>Note : </u>Functions are not case sensitive.

Examples

<OcxControl>.Sqr(5) =   25.

Properties and Methods

Eval-O-Matic supports the following properties and functions.

Properties :

Expression
BoolStyle
QuoteStyle
ErrorEvent
ErrorException
ResultStr
ResultVar

Functions :

AddVariable
GetVarValue
SetVarValue

Expression

Expression	YES	YES	String

Description: This is the expression you are trying to evaluate. It can be any valid expression including any of the built in functions.

Valid expressions follow the standard rules for mathematical expressions. Invalid expressions might include non-mathematical symbols such as “@” or might have open parenthesis, etc.

Example: Evalomatic1.Expression = “5+6-(3*3)”

BoolStyle

BoolStyle	YES	YES	Small Integer

Description: Allows you to set how Boolean expressions are evaluated. Valid values are:
0 = TRUE and FALSE as Boolean variable types
1 = Returns the integer value of 1 for True and 0 for false.
2 = “TRUE” and “FALSE” returned as strings.

Default = 0

Example: Evalomatic1.BoolStyle = 2

QuoteStyle

QuoteStyle	YES	YES	Small Integer

Description: Sets the preferred type of quotes to use in expressions to denote strings. Depending on the language, it might be easier to use the opposite type of quote symbols from the compiler to avoid use of multiple quotes. Valid values are:
0 = double quotes (")
1 = single quotes (')

Default = 0

Example:

```
Evalomatic1.QuoteStyle = 0
Evalomatic1.Expression = """hello"" > ""mello"""
Evalomatic1.QuoteStyle = 1
Evalomatic1.Expression = " 'hello' > 'mello' "
```

ErrorEvent

Type : Boolean

Use : This property determines whether the error event would be fired or not in the case of error while evaluating the expression.

If set to TRUE the error event will be fired.
If set to FALSE the error event will not be fired.

Example :

```
Evalomatic1..ErrorEvent = TRUE
```

ErrorException

ErrorException	YES	YES	Boolean

Description: Determines if an exception is thrown when an exception occurs during evaluation of the expression. If true an exception will be thrown, if false it will not be thrown.

Default = True

Example: Evalomatic1.ErrorException = FALSE

ResultStr

ResultStr	YES	NO	String

Description: Contains the result of the expression if it can be computed, as a string.

Example: Textbox1.Text = Evalomatic1.ResultStr

ResultVar

ResultVar	YES	NO	Variant

Description: Contains the result of the expression if it can be computed, as a variant.

Example:
```
MyIntegerVar = Evalomatic1.ResultVar
MyStringVar = Evalomatic1.ResultVar
```

AddVariable

Name	String	Name of the variable.
Value	Variant	The value to initially set the variable to.
nothing		There is no return value.

Description: AddVariable(ByVal Name as String,
ByVal Var Value as Variant)

It adds a variable so that you can use a variable in an expression. During evaluation, it will substitute the value for the variable. You can change the value of the variable by using SetVariable without changing the expression.

Example: Evalomatic1.AddVariable(“MyVar”, 17)

SetVariable

Name	String	Name of the variable.
Value	Variant	The value to initially set the variable to.
nothing		There is no return value.

Description: SetVariable(ByVal Name as String,
ByVal VarValue as Variant)

It changes the value of a variable that you have already set using AddVariable.

Example: Evalomatic1.SetVariable(“MyVar”, 18)

GetVariable

Name	String	Name of the variable.
	Variant	Returns the current value of the variable.

Description: GetVariable(ByVal Name as String)
Returns the value of an existing variable.

Example: MyVar = Evalomatic1.GetVariable(“MyVar”)

DisplayStyle

DisplayStyle	YES	YES	Integer

Description:	Determines how the calculator is displayed. Options are: 0 = Drop-down combo box. 1 = Embedded in a form. 2 = As a floating window.
Example:	CalcOmatic1.DisplayStyle = 1

CalcStyle

CalcStyle	YES	YES	Integer

Description:	Determines which calculator style is displayed. 0 = Standard 1 = Scientific 2 = Extended Scientific 3 = Financial 4 = Statistical 5 = Café Classic
Example:	CalcOmatic1.CalcStyle = 1

FloatCalcVisible

FloatCalcVisible	YES	YES	Boolean

Description:	If set to be a floating calculator, this will show and hide the calculator.
	True = Visible
	False = Invisible

Example:	CalcOmatic1.FloatCalcVisible = False

FloatTitle

FloatTitle	YES	YES	String

Description:	Allows you to set the text in the title bar of a floating calculator.
Example:	CalcOmatic1.FloatTitle = “My Calc”

CCDropDown()

none
nothing

Description: When in combo-box style, fires when the calculator is dropped down. Allows the programmer to stuff a value into the display, or to do anything else before the user has a chance to use the calculator.

Example:

```
Event CCDropDown()
begin
    CalcOmatic1.Expression = '1+1'
end
```

CCPopUp()

CalcDisplay	string	When in combo box style, the contents of the calculator display just before the user pops the calculator up.
nothing		

Description: When in combo-box style, fires when the calculator is popping up. It allows the programmer to respond when the user is done with the calculator.

Example:

```
Event CCPopUp()
begin
    CalcOmatic1.Text = “$” + CalcDisplay
end
```

Expression

Expression	YES	YES	String

Description: This is the expression you are trying to evaluate. It can be any valid expression including any of the built in functions.

Valid expressions follow the standard rules for mathematical expressions. Invalid expressions might include non-mathematical symbols such as “@” or might have open parenthesis, etc.

Example: CalcOmatic1.Expression = “5+6-(3*3)”

BoolStyle

BoolStyle	YES	YES	Small Integer

Description: Allows you to set how Boolean expressions are evaluated. Valid values are:
0 = TRUE and FALSE as Boolean variable types
1 = Returns the integer value of 1 for True and 0 for false.
2 = “TRUE” and “FALSE” returned as strings.

Default = 0

Example: CalcOmatic1.BoolStyle = 2

QuoteStyle

QuoteStyle	YES	YES	Small Integer

Description: Sets the preferred type of quotes to use in expressions to denote strings. Depending on the language, it might be easier to use the opposite type of quote symbols from the compiler to avoid use of multiple quotes. Valid values are:
0 = double quotes (")
1 = single quotes (')

Default = 0

Example:

```
CalcOmatic1.QuoteStyle = 0
CalcOmatic1.Expression = """hello"" > ""mello"""
CalcOmatic1.QuoteStyle = 1
CalcOmatic1.Expression = " 'hello' > 'mello' "
```

ResultStr

ResultStr	YES	NO	String

Description: Contains the result of the expression if it can be computed, as a string.

Example:
```
Textbox1.Text = CalcOmatic1.ResultStr
```

ResultVar

ResultVar	YES	NO	Variant

Description: Contains the result of the expression if it can be computed, as a variant.

Example:
```
MyIntegerVar = CalcOmatic1.ResultVar
MyStringVar = CalcOmatic1.ResultVar
```

Properties and Methods

Calc-O-Matic supports the following properties and functions.

Properties :

CalcStyle
DisplayStyle
FloatCalcVisible
EmbedBorderStyle
FloatTitle
Expression
ResultStr
ResultVar
BoolStyle
QuoteStyle

Events :

CCDropDown()
 CCPopUp()

Overview

The Eval-O-Matic is a very straight forward control to use. It is invisible at runtime, and has a minimum number of properties and methods. To evaluate a complex mathematical expression, all you have to do is set the expression property, and read from the result property. The Eval-O-Matic automatically parses the string expression, and performs the calculations. It knows all about precedence of operators, and about nesting parenthesis. It even knows over 100 financial, statistical, and mathematical functions- most of which are compatible with MS Excel, so it will be easy for most users to use these functions.

Overview

The Func-O-Matic is a set of functions, all of which can be used by the Eval-O-Matic, but that can be used directly without the overhead of the string parser. These functions can be used two ways - by directly calling them in a DLL, or by using the OCX which is just a very thin wrapper to the DLL, to make calling easier with very little additional overhead.

Overview

The Calc-O-Matic is three controls in one! By changing the style property, you have a combo box, an embeddable calculator, or a floating calculator applet. No matter how you choose to display it, it does the same work. It displays a series of buttons that users can use - just like real calculator buttons - and uses the Eval-O-Matic calculating engine to produce the results. There are 6 different styles of calculators for you to choose from. In each case, users can type complex expressions whether or not the buttons are displayed, giving them maximum power!

UserFunction

Name	String	The name of the unknown function.
InParam	Variant	The parameters for the user to use.
OutParam	Variant	The result to pass back to the EvalOMatic so it can continue computing.

Description:
```
UserFunction(        Name as String,
                     InParam as Variant,
                     ByVal OutParam as Variant)
```
When the EvalOMatic encounters a function that it doesn't know how to handle, it fires this event to allow the programmer to handle it. The EvalOMatic first tries to evaluate anything in the parenthesis that it can, then passes the results to the event via the InParam parameter. If the expression cannot be evaluated, it might pass it in as a string and leave it to the user to interpret the results.

Example:
```
begin
   Evalomatic1.Expression = “MyFunction(2+2,9/3)”
   Result = EvalOMatic1.ResultVar
end

UserFunction(Name, InParam, OutParam)
begin
   ‘This demonstrates a simple user function that
   ‘takes two parameters, performs a calculation,
   ‘the returns the value to the EvalOMatic.

   If name = “MyFunction” then begin
      ‘grab the first variable by taking the string up to
      ‘the first comma
      Var1 = Instr(InParam,1,Pos(InParam,”,”)-1)

      ‘get the second variable by chopping the rest
      ‘of the string
      Var2 = InStr(    InParam,
                       Pos(InParam,”,”)+1,
                       Length(InParam))
      OutParam = Var1 + 2 * Var2
   end
end
```

See Also

ErrorDetect

ErrorDetect

ErrorCode	Long Integer	The number of an error.
ErrorCause	String	Text of the error message.
Noting		

Description: When there is an error during the evaluation of an expression, this event is fired off giving the programmer a chance to handle it. The error codes and causes can be found in a later chapter.

Example:

```
Event ErrorDetect(ErrorCode, ErrorCause)
begin
    MessageBox(ErrorCause)
end
```

See Also

UserFunction

Distribution Files

If you distribute an application that uses the Eval-O-Matic, Calc-O-Matic, or Func-O-Matic, you will need to include the following files:

ccEval.ocx
ccFunc.dll

In addition, you will need to make sure the following Microsoft files are on the user's system, and may need to distribute the following shared files as well.

MFC40.DLL
MSVCRT40.dll
OLEPRO32.DLL
REGSVR32.EXE

Angle Unit

AngleUnit	YES	YES	Integer

Description: Selects the units that trigonometric functions use. Valid options are:
0 = Radians
1 = Degrees
2= Gradians

Example: EvalOMatic1.AngleUnit = 1

ANGLEUNIT

AngleUnit	YES	YES	Integer

Description: Selects the units that trigonometric functions use. Valid options are:
0 = Radians
1 = Degrees
2= Gradians

Example: FuncOMatic1.AngleUnit = 1

ANGLEUNIT

AngleUnit	YES	YES	Integer

Description: Selects the units that trigonometric functions use. Valid options are:
0 = Radians
1 = Degrees
2= Gradians

Example: CalcOMatic1.AngleUnit = 1

Registration Card

To register a product, or order additional products:

By Mail:
Component Café
PO Box 542269
Houston, TX 77254

By Phone:
1-888-889-5565

Name

Company Name

Title

Address

City State Zip

Country

( ) Phone ( ) Fax

email

Where would you like updates sent: Email Snail Mail None

What type of programmer are you?

- Hobbyist
- Independent Consultant
- Staff Programmer for Large Company
- Programmer / "wears many hats" for smaller company
- Other

Where did you hear about us?

Did you have to clear this purchase with another department?

I'd like to buy:

Control	**Quantity**		**Price**	**Total**
Eval-O-Matic		X	189	
Calc-O-Matic		X	189	
Func-O-Matic		X	79	
All Three!		X	249	
Upgrade from Func to all three		X	179	
Upgrade from Calc or Eval to all three		X	89	
			Subtotal:	
			Residents of Texas, add 8% sales tax:	
			Shipping and Handling, add $7 for the first item and $3 for each additional item:	

TOTAL:

Method of Payment (check one):
Check or Money Order in US dollars (checks from US banks only)
Mastercard
Visa

credit card number expiration date

name on card (please print in all caps, block letters)

I authorize Component Café to charge my credit card:

signature

Starting Install

Insert the first floppy into a 1.44 floppy drive. Run SETUP.EXE. An Installsheild® wizard will walk you through the options.

All three components documented in this manual will be installed. We’ve found that users prefer to limit the number of files they have to distribute.   Controls that you have not chosen to license will be available to you in a demo-mode.

Visual Basic 4.0

1. Load Visual Basic 4.0, 32 bit version.
2. Select Tools, Custom Controls.
3. Looks for “Component Café EvalOMatic”, and check its box.
4. Select OK.

Delphi 2.0

1. Load Delphi. If already running, close any open projects.
2. Select Component, Install from the menu.
3. Click the button labeled “OCX.”
1. Scroll down the list of controls. Look for the one labeled “Component Café EvalOMatic.” Select it.
2. Click OK. And OK again. The library will rebuild itself and the components will appear on the OCX palette.

Microsoft Visual C++ 4.0

1. Load VC++. If already running, close any open projects.
2. Select Insert, Component from the menu.
3. Select the “OCX” tab.
1. You will find the controls “EvalOMatic”, “FuncOMatic”, and “CalcOMatic.” Select the ones you wish to use, and click the “Insert” button.
1. Click “Close” to finish.

Common Questions:

Q: What happens if I compile a project and don’t have a license?
A: A friendly little box pops up telling people about our controls during run-time when they first load the application.

Q: What if I have a license for one control, but not the others that are included in the OCX file?
A: If you use only the controls you are licensed for, there will be no messages displayed during run-time. The control is smart enough to tell which of the controls you have a license for and which ones you do not.

Q: Do I have to distribute the license file in order to make sure the message boxes do not pop-up?
A: No. In fact, you are not supposed to distribute the license file at any time. The signal to the control whether or not to show a message during run-time happens during the compile process. Distributing the license file would make no difference what-so-ever.

Q: If I want to obtain a license, what do I have to do?
A: Just call us! You can get a license over the phone!

Q: What else does registering do for me?
A: If you actually register with us (i.e., send in registrations card or contact us via phone, web, or email to register your product), you’ll automatically receive all patches and fixes to the current version of the product, as well as any special offers that might become available to registered users.

General License Agreement

The full license agreement can be found at the end of the manual. The highlights you should keep in mind are:

- We encourage you to distribute the OCX files to any programmers that might find them useful, as long as you do not also distribute the license (.lic) file.

Our license is pretty liberal. You should own as many licenses as you have developers using the product. You may keep license files on multiple PC's, for instance a lap-top, an office machine and your home machine, as long as there is not more than one developer using it at a time. If there is a chance that two people will be developing at the same time, then two licenses are required.

EmbedBorderStyle

EmbedBorderStyle	YES	YES	Integer

Description: If used as an embedded control, this determines the border-style.
Options are:
0 = No border.
1 = Thin.
2 = Sunken.
3 = Raised.
4 = Etched.
5 = Bump.

Example: CalcOMatic1.EmbedBorderStyle = 1

ErrorException

ErrorException	YES	YES	Boolean

Description:	Determines if an exception is thrown when an exception occurs during evaluation of the expression. If true an exception will be thrown, if false it will not be thrown.
	Default = True
Example:	CalcoMatic1.ErrorException = FALSE

Calculate

none	
nothing	There is no return value.

Description: Forces the calculator to calculate the expression and display the new results - acts as though the user pressed the equals button.

Example: CalcOMatic1.Calculate

OnCalculate

none
nothing

Description: Fires just after the user presses the equals sign.

Example:
```
Event OnCalculate
begin
    LastExpression = CalcOMatic1..Text
end
```

CalcText

CalcText	YES	YES	String

Description: Reads and sets the text displayed. For the combo box, it reads and sets the text in the combo box, but not in the calculator, whereas setting the expression will change the text in the calculator, but not affect the text in the combo box. For the embedded and floating forms, this acts like expression.

Example: CalcOMatic1.CalcText = “1+1”

User Function

Name	String	The name of the unknown function.
InParam	Variant	The parameters for the user to use.
OutParam	Variant	The result to pass back to the EvalOMatic so it can continue computing.

Description:

```
UserFunction(           Name as String,
                        InParam as Variant,
                        ByVal OutParam as Variant)
```

When the EvalOMatic encounters a function that it doesn't know how to handle, it fires this event to allow the programmer to handle it. The EvalOMatic first tries to evaluate anything in the parenthesis that it can, then passes the results to the event via the InParam parameter. If the expression cannot be evaluated, it might pass it in as a string and leave it to the user to interpret the results.

Example:

```
begin
    Evalomatic1.Expression = “MyFunction(2+2,9/3)”
    Result = EvalOMatic1.ResultVar
end

UserFunction(Name, InParam, OutParam)
begin
    ‘This demonstrates a simple user function that
    ‘takes two parameters, performs a calculation,
    ‘the returns the value to the EvalOMatic.

    If name = “MyFunction” then begin
        ‘grab the first variable by taking the string up to
        ‘the first comma
        Var1 = Instr(InParam,1,Pos(InParam,”,”)-1)

        ‘get the second variable by chopping the rest
        ‘of the string
        Var2 = InStr(     InParam,
                          Pos(InParam,”,”)+1,
                          Length(InParam))
        OutParam = Var1 + 2 * Var2
    end
end
```

CMFEET

Usage

<OcxControl>. CMFEET (number).

Description

Converts CentiMeter to Feet.

Note : Functions are not case sensitive.

Example

CMFEET(30) gives 0.984252

CMINCH

Usage

<OcxControl>. CMINCH (number).

Description

Converts CentiMeter   to Inch.

Note : Functions are not case sensitive.

Example

CMINCH(5) gives 1.9685

CTOF

Usage

<OcxControl>. CTOF(number).

Description

Converts Centigrade to Farenheit.

Note : Functions are not case sensitive.

Example

CTOF(35) gives 95

CTOK

Usage

<OcxControl>. CTOK(number).

Description

Converts Centigrade to Kelvin.

Note : Functions are not case sensitive.

Example

CTOK(75) gives 167

FEETCM

Usage

<OcxControl>. FEETCM(number).

Description

Converts Feet to Centimeter.

Note : Functions are not case sensitive.

Example

FEETCM(1) gives 30.48

FEETM

Usage

<OcxControl>. FEETTM (number).

Description

Converts Feet to Metre.

Note : Functions are not case sensitive.

Example

FEETM(200) gives 60.96

FTOC

Usage

<OcxControl>. FTOC (number).

Description

Converts Farenheit to Centigrade

Note : Functions are not case sensitive.

Example

FTOC(200) gives 93.33333

GALLTR

Usage

<OcxControl>. GALLTR (number).

Description

Converts Gallon to Litre.

Note : Functions are not case sensitive.

Example

GALLTR(10) gives 45.4609

INCM

Usage

<OcxControl>. INCM(number).

Description

Converts Inch to Centimetre.

Note : Functions are not case sensitive.

Example

INCM(20) gives 50.8

KGPOUND

Usage

<OcxControl>. KGPOUND(number).

Description

Converts Kilogram to Pound.

Note : Functions are not case sensitive.

Example

KGPOUND(100) gives 220

KMMILE

Usage

<OcxControl>. KMMILE(number).

Description

Converts Kilometre to Mile

Note : Functions are not case sensitive.

Example

KMMILE(8) gives 4.97104

LTRGAL

Usage

<OcxControl>. LTRGAL(number).

Description

Converts Litre to Gallon

Note : Functions are not case sensitive.

Example

GALLTR(10) gives 45.4609

MFEET

Usage

<OcxControl>. MFEET (number).

Description

Converts Metre to Feet.

Note : Functions are not case sensitive.

Example

MFEET(12) gives 39.37007.

MILEKM

Usage

<OcxControl>. MFEET (number).

Description

Converts Mile to Kilometre.

Note : Functions are not case sensitive.

Example

MILEKM(100) gives 160.93

MLOZ

Usage

<OcxControl>. MLOZ(number).

Description

Converts Milli to Ounce

Note : Functions are not case sensitive.

Example

MLOZ(202) gives 7.1104

OZML

Usage

<OcxControl>. OZML(number).

Description

Converts Ounce to Milli.

Note : Functions are not case sensitive.

Example

OZML(100) gives 2840

POUNDKG

Usage

<OcxControl>. POUNDKG(number).

Description

Converts Pound to Kilogram.

Note : Functions are not case sensitive.

Example

POUNDKG(20) gives 9.0909

SQFEETSQM

Usage

<OcxControl>. SQFEETSQM(number).

Description

Converts SquareFeet to SquareMetre.

Note : Functions are not case sensitive.

Example

SQFEETSQM(100) gives 9.290304

SQMSQFEET

Usage

<OcxControl>. SQMSQFEET (number).

Description

Converts SquareMetre to SquareFeet.

Note : Functions are not case sensitive.

Example

SQMSQFEET(10) gives 107.639

KTOC

Usage

KTOC(number) where number is the argument to be converted to kelvin

Description

Converts Centigrade to Kelvin.

Example

KTOC(300) gives 4.94065

VB Example

See Also

CTOK , CTOF , FTOC

KTOC

Usage

    <OcxControl>. KTOC(number).

Description

    Converts Farenheit   to Celsius.

Note : Functions are not case sensitive.

Example

KTOC(300) gives 4.94065

MIRR

Returns the modified internal rate of return for a series of periodic cash flows. MIRR considers both the cost of the investment and the interest received on reinvestment of cash.

Syntax
<OcxControl>.MIRR(values, finance_rate, reinvest_rate)

Values is an array or a reference to cells that contain numbers. These numbers represent a series of payments (negative values) and income (positive values) occurring at regular periods.

Values must contain at least one positive value and one negative value to calculate the modified internal rate of return. Otherwise, MIRR returns the #DIV/0! error value.
Finance_rate is the interest rate you pay on the money used in the cash flows.
Reinvest_rate is the interest rate you receive on the cash flows as you reinvest them.

Note : Functions are not case sensitive.

Examples

Suppose you're a commercial fisherman just completing your fifth year of operation. Five years ago, you borrowed $120,000 at 10 percent annual interest to purchase a boat. Your catches have yielded $39,000, $30,000, $21,000, $37,000, and $46,000. During these years you reinvested your profits, earning 12% annually. In a worksheet, your loan amount is entered as -$120,000 To calculate the investment's modified rate of return after five years:

<OcxControl>.MIRR(39,000, 30,000, 21,000, 37,000, 46,000 10%, 12%) equals 12.61%

IF

Returns one value if logical_test evaluates to TRUE and another value if it evaluates to FALSE.

There are two syntax forms of the IF function. Syntax 1 can be used on worksheets and macro sheets. Syntax 2 can only be used on macro sheets in conjunction with the ELSE, ELSE.IF, and END.IF functions.
Use IF to conduct conditional tests on values and formulas and to branch based on the result of that test. The outcome of the test determines the value returned by the IF function.

Syntax 1

Note : Functions are not case sensitive.

Remarks

You could use the following nested IF function:

Examples

```
<OcxControl>.if(Average>89,"A",if(Average>79,"B",
<OcxControl>.if(Average>69,"C",if(Average>59,"D","F"))))
```

In the preceding example, the second IF statement is also the value_if_false argument to the first IF statement. Similarly, the third IF statement is the value_if_false argument to the second IF statement. For example, if the first logical_test (Average>89) is TRUE, "A" is returned. If the first logical_test is FALSE, the second IF statement is evaluated, and so on.

QUICKSORT

General Description

To sort the given array in the ascending order

Usage

QUICKSORT(number1,number2,number3….)

Example

QUICKSORT(1,3,7,5) gives 1,3,5,7.

ADDVARIABLE

Name	String	Name of the variable.
Value	Variant	The value to initially set the variable to.
nothing		There is no return value.

Description: AddVariable(ByVal Name as String,
ByVal Var Value as Variant)

It adds a variable so that you can use a variable in an expression. During evaluation, it will substitute the value for the variable. You can change the value of the variable by using SetVariable without changing the expression.

Example: CalcOmatic1.AddVariable(“MyVar”, 17)

SETVARIABLE

Name	String	Name of the variable.
Value	Variant	The value to initially set the variable to.
nothing		There is no return value.

Description: SetVariable(ByVal Name as String,
ByVal VarValue as Variant)
It changes the value of a variable that you have already set using AddVariable.

Example: CalcOmatic1.SetVariable(“MyVar”, 18)

GETVARIABLE

Name	String	Name of the variable.
	Variant	Returns the current value of the variable.

Description: GetVariable(ByVal Name as String)
Returns the value of an existing variable.

Example: MyVar = CalcOmatic1.GetVariable(“MyVar”)

ERRORDETECT

ErrorCode	Long Integer	The number of an error.
ErrorCause	String	Text of the error message.
Noting		

Description: When there is an error during the evaluation of an expression, this event is fired off giving the programmer a chance to handle it. The error codes and causes can be found in a later chapter.

Example:
```
Event ErrorDetect(ErrorCode, ErrorCause)
begin
   MessageBox(ErrorCause)
end
```

See Also

UserFunction

COMPLEXSUB

Adds two complex numbers together.They are expressed in terms of their real and imaginary parts: a + ib where a is the square root of -1, a and b are both
real numbers.
Syntax

<OcxControl>.ComplexSub(a1,b1,a2,b2,a3,b3)

The function works like this:

a3 = (a1 - a2)
b3 = (b1 - b2)

Note : Functions are not case sensitive.

Example

For two complex numbers: (2 + 3i) + (4 + 7i)
<OcxControl>.ComplexSub(2,3,4,7,A,B);
A = -2
B = -4.