Appendix A Functions

This appendix provides a complete alphabetical reference to the functions.

ABS

Description	Returns the absolute value of a number.
Syntax	**ABS***(number)*
	number is any integer.
See Also	**SIGN** function
Examples	`ABS(-1)` returns 1
	`ABS (1)` returns 1

ACOS

Description	Returns the arc cosine of a number.
Syntax	**ACOS**(*number*)
	number is the cosine of the angle. The cosine can range from 1 to -1.
Remarks	The resulting angle is returned in radians (from 0 to π).

Examples	`ACOS(.5) returns 1.05`
	`ACOS(.2) returns 1.77`

AND

Description	Returns True if all arguments are true; returns False if at least one argument is false.
Syntax	**AND**(*logical_list*)
	logical_list is a list of conditions separated by commas. You can include as many as 30 conditions in the list.
See Also	**IF**, **NOT**, and **OR** functions
Examples	`AND(1+1=2, 5+5=10)` returns True because both arguments are true.

AVERAGE

Description	Returns the average of the supplied numbers. The result of **AVERAGE** is also known as the arithmetic mean.
Syntax	**AVERAGE***(number_list)*
	number_list is a list of numbers separated by commas. As many as 30 numbers can be included in the list.
See Also	**MIN** and **MAX** functions
Examples	`AVERAGE(5, 6, 8, 14)` returns 8,25

CEILING

Description	Rounds a number up to the nearest multiple of a specified significance.
Syntax	**CEILING**(*number, significance*)
	number is the value to round.
	significance is the multiple to which to round.
Remarks	Regardless of the sign of the number, the value is rounded up, away from zero.
	If *number* is an exact multiple of *significance,* no rounding occurs.
See Also	**EVEN, FLOOR, INT, ODD, ROUND,** and **TRUNC** functions

	Examples	`CEILING(1.23459, .05)` returns 1.25
		`CEILING(-148.24, -2)` returns-150
EVEN	**Description**	Rounds the specified number up to the nearest even integer.
	Syntax	**EVEN***(number)*
		number is any number, a formula that evaluates to a number, or a reference to a cell that contains a number.
	See Also	**CEILING, FLOOR, INT, ODD, ROUND,** and **TRUNC** functions
	Examples	`EVEN(2.5)` returns 4
		`EVEN(2030.45)` returns 2032
EXP	**Description**	Returns e raised to the specified power. The constant e is 2.71828182845904 (the base of the natural logarithm).
	Syntax	**EXP***(number)*
		number is any number as the exponent.
	See Also	**LN** and **LOG** functions
	Examples	`EXP(2.5)` returns 12. 1 8
		`EXP(3)` returns 20.09
FLOOR	**Description**	Rounds a number down to the nearest multiple of a specified significance.
	Syntax	**FLOOR**(*number, significance*)
		number is the value to round.
		significance is the multiple to which to round.
	Remarks	Regardless of the sign of the number, the value is rounded down, toward zero. If *number* is an exact multiple of *significance,* no rounding occurs.
	See Also	**CEILING, EVEN, INT, ODD, ROUND**, and **TRUNC** functions
	Examples	`FLOOR(1.23459, .05)` returns 1.2
		`FLOOR(-148.24, -2)` returns –148
HOUR	**Description**	Returns the hour component of the specified time in 24-hour format.
	Syntax	**HOUR**(*serial_number*)
		serial_number is the time as a serial number. The decimal portion of the number represents time as a fraction of the day.
	Remarks	The result is an integer ranging from 0 (12:00 AM) to 23 (11:00 PM).
	See Also	**MINUTE** function
	Examples	`HOUR(34259.4)` returns 9
		`HOUR(34619.976)` returns 23
IF	**Description**	Tests the condition and returns the specified value.
	Syntax	**IF**(*condition, true_value, false_value*)
		condition is any logical expression.
		true_value is the value to be returned if *condition* evaluates to True.
		false_value is the value to be returned if *condition* evaluates to False.
	See Also	**AND, NOT**, and **OR** functions

Example to show in a block the result of *a*/*b* only if *a* is greater than 500 and *b* is different from 0, the formula will be:

`If(AND(a > 500, b <> 0); a/b; 0)`

INT

Description Rounds the supplied number down to the nearest integer.

Syntax **INT**(*number*)

number is any real number.

See Also **CEILING, FLOOR, MOD, ROUND**, and **TRUNC** functions

Examples `INT(10.99)` returns 10

`INT(–10.99)` returns – 11

IPMT

Description Returns the interest payment of an annuity for a given period, based on regular payments and a fixed periodic interest rate.

Syntax **IPMT**(*interest, per, nper, pv, [fv], [type]*)

interest is the fixed periodic interest rate.

per is the period for which to return the interest payment. This number must be between 1 and *nper.*

nper is the number of payments.

pv is the present value, or the lump sum amount the annuity is currently worth.

fv is the future value, or the value after all payments are made. If this argument is omitted, the future value is assumed to be 0.

type indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.

Remarks The units used for *interest* must match those used for *nper.* For example, if the annuity has an 8% annual interest rate over a period of 5 years, specify 8%/12 for *interest* and 5*12 for *nper*.

Cash paid out, such as a payment, is shown as a negative number. Cash received, such as a dividend check, is shown as a positive number.

See Also **PMT, PPMT,** and **RATE** functions

Examples `IPMT(8%/12, 2, 48, 18000)` returns - 117.87

`IPMT(8%/12, 2, 48, 18000, 0, 1)` returns -117.09

LN

Description Returns the natural logarithm (based on the constant e) of a number.

Syntax **LN**(*number*)

number is any positive real number.

Remarks **LN** is the inverse of the EXP function.

See Also **EXP, LOG**, and **LOG10** functions

Examples `LN(12.18)` returns 2.50

`LN(20.09)` returns 3.00

LOG

Description Returns the logarithm of a number to the specified base.

Syntax **LOG**(*number, [base]*)

number is any positive real number.

base is the base of the logarithm. Omitting this argument assumes a base of 10.

See Also **EXP, LN,** and **LOG10** functions

	Examples	`LOG(1)` returns 0
		`LOG(10)` returns 1
LOG10	**Description**	Returns the base-10 logarithm of a number.
	Syntax	**LOG10***(number)*
		number is any positive real number.
	See Also	**EXP, LN**, and **LOG** functions
	Examples	`LOG10(260)` returns 9.41
		`LOG10(100)` returns 2
MAX	**Description**	Returns the largest value in the specified list of numbers.
	Syntax	**MAX**(*number_list*)
		number_list is a list of as many as 30 numbers, separated by commas.
	See Also	**AVERAGE** and **MIN** functions
	Examples	`MAX(50, 100, 150, 500, 200)` returns 500
MIN	**Description**	Returns the smallest value in the specified list of numbers.
	Syntax	**MIN**(*number_list*)
		number_list is a list of as many as 30 numbers, separated by commas.
	See Also	**AVERAGE** and **MAX** functions
	Examples	`MIN(50, 100, 150, 500, 200)` returns 50
MINUTE	**Description**	Returns the minute that corresponds to the supplied date.
	Syntax	**MINUTE**(*serial_number*)
		serial_number is the time as a serial number. The decimal portion of the number represents time as a fraction of the day.
	Remarks	The result is an integer ranging from 0 to 59.
	See Also	**HOUR** function
	Examples	`MINUTE(34506.4)` returns 36
		`MINUTE(34399.925) returns 48`
MOD	**Description**	Returns the remainder after dividing a number by a specified divisor.
	Syntax	**MOD**(*number, divisor*)
		number is any number.
		divisor is any non-zero number.
	See Also	**INT, ROUND**, and **TRUNC** functions
	Examples	`MOD(–23, 3)` returns 1
		`MOD(–23, -3)` returns –2
NOT	**Description**	Returns a logical value that is the opposite of its value.
	Syntax	**NOT***(logical)*
		logical is an expression that returns a logical value (such as, True or False).
	Remarks	*If logical* is false, **NOT** returns True. Conversely, if *logical* is true, **NOT** returns False.
	See Also	**AND, IF**, and **OR** functions

Examples `NOT(TRUE())` returns False

`NOT(2*6 = 12)` returns False

NPER

Description Returns the number of periods of an investment based on regular periodic payments and a fixed interest rate.

Syntax **NPER**(*interest, pmt, pf, [fv], [type]*)

interest is the fixed interest rate.

pmt is the fixed payment made each period. Generally*, pmt* includes the principle and interest, not taxes or other fees.

pf is the present value, the lump-sum amount that a series of future payments is currently worth.

fv is the future value, the balance to attain after the final payment. Omitting this argument assumes a future balance of 0.

type indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.

See Also **IPMT, PMT, PPMT, PV**, and **RATE** functions

Examples `NPER(12%/12, -350, -300, 16000, 1)` returns 36.67

`NPER(1%, -350, -300, 16000)` returns 36.9%

NPV

Description Returns the net present value of an investment based on a series of periodic payments and a discount rate.

Syntax **NPV**(*discount_rate, value_list*)

discount_rate is the rate of discount for one period.

value_list is a fist of as many as 29 arguments or a reference to a range that contains values that represent payments and income.

During calculation, **NPV** uses the order in which the values appear to determine the order of cash flow.

Remarks The time span **NPV** uses for calculation begins one period before the first cash flow date and ends when the last cash flow payment is made. This function is based on future cash flows. When your first cash flow occurs at the beginning of the first period, the first value must be added to the **NPV** result, not supplied as a value in *value_list.*

See Also **PV** function

Example `NPV(8%, -12000, 3000, 3000, 3000, 7000)` returns 811.57

ODD

Description Rounds the specified number up to the nearest odd integer.

Syntax **ODD***(number)*

number is any number or a formula that evaluates to a number.

See Also **CEILING, EVEN, FLOOR, INT, ROUND,** and **TRUNC** functions

Examples `OOD(3.5)` returns 5

`ODD(6)` returns 7

OR

Description Returns True if at least one of a series of logical arguments is true

Syntax **OR**(*logical_list*)

logical_list is a list of conditions separated by commas. You can include as many as 30 conditions in the list.

See Also **AND, IF,** and **NOT** functions

Example `OR(1 + 1 = 1, 5 + 5 = 10)` returns True because one of the arguments is true.

PMT

Description Returns the periodic payment of an annuity, based on regular payments and a fixed periodic interest rate.

Syntax **PMT**(*interest, nper, pv, [fv], [type])*

interest is the fixed periodic interest rate.

nper is the number of periods in the annuity.

pv is the present value, or the amount the annuity is currently worth.

fv is the future value, or the amount the annuity will be worth. When you omit this argument, a future value of 0 is assumed.

type indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.

Remarks **PMT** returns only the principal and interest payment, it does not include taxes or other fees.

The units used for interest must match those used for *nper* For example, if the annuity has an 8% annual interest rate over a period of 5 years, specify 8%/12 for *interest* and 5*12 for *nper.*

Cash paid out, such as a payment, is shown as a negative number. Cash received, such as a dividend check, is shown as a positive number.

See Also **IPMT, NPER, PPMT, PV,** and **RATE** functions

Examples `PMT(8%/12, 48, 18000)` returns -439.43

`PMT(8%/12, 48, 18000, 0, 1)` returns -436.52

PPMT

Description Returns the principle paid on an annuity for a given period.

Syntax **PPMT**(*interest, per, nper, pv, [fv], [type])*

interest is the fixed periodic interest rate.

per is the period for which to renew the principle.

nper is the number of periods in the annuity.

pv is the present value, or the amount the annuity is currently worth.

fv is the future value, or the amount the annuity will be worth. When you omit this argument, a future value of 0 is assumed.

type indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.

Remarks The units used for *interest* must match those used for *nper.* For example, if the annuity has an 8% annual interest rate over a period of 5 years, specify 8%/12 for *interest* and 5*12 for *nper.*

See Also **IPMT, NPER, PMT, PV**, and **RATE** functions

Examples `PPMT(8%/12, 2, 48, 18000)` returns -321,56

`PPMT(8%/12, 2, 48, 18000, 0, 1)` returns -319.43

PV

Description Returns the present value of an annuity, considering a series of constant payments made over a regular payment period.

Syntax **PV**(*interest, nper, pmt, [fv], [type])*

interest is the fixed periodic interest rate.

nper is the number of payment periods in the investment.

pmt is the fixed payment made each period.

fv is the future value, or the amount the annuity will be worth. When you omit this argument, a future value of 0 is assumed.

type indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.

Remarks The units used for *interest* must match those used for *nper.* For example, if the annuity has an 8% annual interest rate over a period of 5 years, specify 8%/12 for *interest* and 5*12 for *nper.*

Cash paid out, such as a payment, is shown as a negative number. Cash received, such as a dividend check, is shown as a positive number.

See Also **IPMT, NPER, PMT, PPMT**, and **RATE** functions

Examples `PV(8%/12, 48, 439.43)` returns -17999.89

`PV(8%/12, 48, -439.43)` returns 17999.89

RAND

Description Returns a number selected randomly from a uniform distribution greater than or equal to 0 and less than 1.

Syntax **RAND**()

Remarks Although **RAND** does not use arguments, you must supply the empty parentheses to correctly reference the function.

Example `RAND()*10` returns a random number greater than or equal to 0 and less than 10.

RATE

Description Returns the interest rate per period of an annuity, given a series of constant cash payments made over a regular payment period.

Syntax **RATE**(*nper, pmt, pv, [fv], [type], [guess])*

nper is the number of periods in the annuity.

pmt is the fixed payment made each period. Generally, *pmt* includes only principle and interest, not taxes or other fees.

pv is the present value of the annuity.

fv is the future value, or the amount the annuity will be worth. When you omit this argument, a future value of 0 is assumed.

type indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.

guess is your estimate of the interest rate. If no argument is supplied, a value of. 1 (10%) is assumed.

Remarks **RATE** is calculated iteratively, cycling through the calculation until the result is accurate to .00001 percent.

See Also **IPMT, NPER, PMT, PPMT**, and **PV** functions

Example `RATE(48, -439.43, 18000)` returns .0067 (rounded to 4 decimals), which is the monthly interest rate. The annual interest rate (.0067 multiplied by 12) is 8%.

ROUND

Description Rounds the given number to the supplied number of decimal places,

Syntax **ROUND**(*number, precision*)

number is any value.

precision is the number of decimal places to which *number* is rounded.

When a negative precision is used, the digits to the right of the decimal point are dropped and the absolute number of significant digits specified by *precision* are replaced with zeros.

If precision is 0, number is rounded to the nearest integer.

See Also **CEILING, FLOOR, INT, MOD**, and **TRUNC** functions

Examples `ROUND(123.456, 2)` returns 123.46

`ROUND(9899.435, -2)` returns 9900

SIGN

Description Determines the sign of the specified number.

Syntax **SIGN***(number)*

number is any number.

Remarks **SIGN** returns 1 if the specified number is positive, -1 if it is negative, and 0 if it is 0.

See Also **ABS** function

Examples `SIGN(-123)` returns –1

`SIGN(123)` returns 1

SQRT

Description Returns the square root of the specified number.

Syntax **SQRT***(number)*

number is any positive number.

Examples `SQRT(9)` returns 3

`SQRT(2.5)` returns 1.58

STDEV

Description Returns the standard deviation of a population based on a sample of supplied values. The standard deviation of a population represents an average of deviations from the population mean within a list of values.

Syntax **STDEV**(*number_list*)

number_list is a list of as many as 30 numbers, separated by commas.

See Also **STDEVP, VAR**, and **VARP** functions

Example `STDEV(4.0, 3.0, 3.0, 3.5, 2.5, 4.0, 3.5)` returns .56

STDEVP

Description Returns the standard deviation of a population based on an entire population of values. The standard deviation of a population represents an average of deviations from the population mean within a list of values.

Syntax **STDEVP***(number_list)*

number_list is a list of as many as 30 numbers, separated by commas.

See Also **STDEV, VAR**, and **VARP** functions

Example `STDEVP(4.0, 3.0, 3.0, 3.5, 2.5, 4.0, 3.5)` returns .52

SYD

Description Returns the depreciation of an asset for a specified period using the sum-of-years method. This depreciation method uses an accelerated rate, where the greatest depreciation occurs early in the useful life of the asset.

Syntax **SYD**(*cost, salvage, life, per)*

cost is the initial cost of the asset.

salvage is the salvage value of the asset.

life is the number of periods in the useful life of the asset.

period is the period for which to calculate the depreciation. The time units used to determine *period* and *life* must match.

See Also **VDB** function

Example `SYD(10000, 1000, 7, 3)` returns 1607.14

TRUNC

Description Truncates the given number to an integer.

Syntax **TRUNC**(*number, [precision])*

number is any value.

precision is the number of decimal places allowed in the truncated number. Omitting this argument assumes a precision of 0.

Remarks **TRUNC** removes the fractional part of a number to the specified precision without rounding the number.

See Also **CEILING, FLOOR, INT, MOD**, and **ROUND** functions

Examples `TRUNC(123.456, 2)` returns 123.45

`TRUNC(9899.435, -2)` returns 9800

VAR

Description Returns the variance of a population based on a sample of values.

Syntax **VAR**(*number_list*)

number_list is a list of as many as 30 numbers, separated by commas.

See Also **STDEV, STDEVP,** and **VARP** functions

Example `VAR(4.0, 3.0, 3.0, 3.5, 2.5, 4.0, 3.5)` returns .31

VARP

Description Returns the variance of a population based on an entire population of values.

Syntax **VARP**(number_list)

number_list is a list of as many as 30 numbers, separated by commas. The list can contain numbers or a reference to a range that contains numbers.

See Also **STDEV, STDEVP**, and **VAR** functions

Example `VARP(4.0. 3.0. 3.0, 3.5, 2.5, 4.0, 3.5)` returns .27

VDB

Description Returns the depreciation of an asset for a specified period using a variable method of depreciation.

Syntax	**VDB**(*cost, salvage, life, start_period, end_period, [factor], [method])*
	cost is the initial cost of the asset.
	salvage is the salvage value of the asset.
	life is the number of periods in the useful life of the asset.
	start_period is the beginning period for which to calculate the depreciation. The time units used to determine *start_period and life* must match.
	end_period is the ending period for which to calculate the depreciation. The time units used to determine *end_period and* life must match.
	factor is the rate at which the balance declines. Omitting this argument assumes a default of 2, which is the double-declining balance factor.
	method is a logical value that determines if you want to switch to straight-line depreciation when depreciation is greater than the declining balance calculation. Use True to maintain declining balance calculation; use False or omit the argument to switch to straight-line depreciation calculation.
See Also	**SYD** functions
Example	`VDB (10000, 1000, 7, 3, 4)` returns 1041.23