Function reference

1 Data Type

➢ “”	String
➢ ‘’	Variable name
➢ 0x12	Hexadecimal long integer
➢ 1L	Long integer
➢ \s	A character that invokes an alternative interpretation on special characters in a string
➢ date()	Convert a string or integer to date
➢ datetime()	Convert the string or long integer to date time
➢ decimal()	Convert to big decimal number
➢ float()	Convert to double number
➢ false	False value
➢ ifdate()	Judge if the parameter is a date or a date time
➢ ifnumber()	Judge if the parameter is a number
➢ ifstring()	Judge if the parameter is a string
➢ iftime()	Judge if the parameter is a time
➢ int()	Convert to integer
➢ long()	Convert to long integer
➢ number()	Convert to real number
➢ null	Null value
➢ string()	Convert to string
➢ time()	Convert the string or integer to time data
➢ true	True value

2 Operator and Function

➢ ${ }	Macro
➢ %\	Get the remainder and the integer value
➢ &	Generate a new sequence by merging two sequences
➢ &&,||,!	Logic operation
➢ (x_1,x_2,…,x_k)	Batch computation, to compute a series of expressions one by one in an automated fashion, and return the result of the last expression
➢ *	Generate a new sequence by duplicating members of a sequence
➢ ++,--,**,//,%%,\\	Generate a new sequence by Alignment Arithmetic Operation between two sequences which are of the same length, such as aligning add, aligning subtract, aligning multiply and so on
➢ +,-,*,/	The four fundamental operations of arithmetic
➢ -a	Opposite number
➢ ==,!=,<,>,<=,>=	Comparison operation
➢ \	Generate a new sequence by subtracting members from a sequence
➢ ^	Generate a new sequence which is composed of common members from two sequences

- a=x — Assign the result of an expression to a variable and return the result of the expression
- a?=x — Compound assignment computation
- case() — According to the various results of judge expressions, return various values
- cmp() — Compare the value of two expressions or two sequences
- eq() — Judge if a sequence can be generated by swapping the positions of the members of another sequence
- eval() — Dynamically parse and compute the expression
- f@o(…) — Introduce the common rules of functions
- if() — According to the various results of boolean expressions, return various values
- in() — Judge if the Parameter 1 is between the Parameter 2 and Parameter 3
- s_1+s_2 — Join two or more strings end-to-end
- | — Concatenate two sequences so as to generate a new sequence

3 Mathematic Function

- abs() — Absolute value
- acos() — Arc cosine value
- acosh() — Return the inverse hyperbolic cosine
- and() — Perform bitwise AND operation on integers
- asin() — Arc sine value
- asinh() — Return the inverse hyperbolic sine
- atan() — Arc tangent value
- atanh() — Return the inverse hyperbolic tangent
- bits() — Convert to decimal numbers
- ceil() — Truncate the data at the specified position, and carry the remaining part if any
- combin() — Return the number of combinations
- cos() — Cosine value
- cosh() — Return the hyperbolic cosine
- exp() — Powers of e
- fact() — Factorial
- floor() — Truncate the data at the specified positions, and reject all the remaining part if any
- gcd() — Return the greatest common divisor
- lcm() — Return the least common multiple
- ifnumber() — If the parameter is a number
- lg() — Logarithm with 10 as its base
- ln() — Natural logarithm
- not() — On integers, perform bitwise NOT operation to get the logical negation on each bit

- or() Perform bitwise OR operation on integers
- pi() Circumference ratio
- permut() Return the number of permutations
- power() Powers of a numeric value
- product() Get the product
- rand() Random value
- rgb() Convert the red, green, blue, and transparency value to the corresponding color value
- round() Truncate the data at the specified position, and round off the remaining part
- shift() Perform shift operation
- sign() Judge whether the parameter is positive, negative or 0
- sin() Sine value
- sinh() Judge whether the parameter is a positive or negative number or a 0
- sqrt() Square root
- tan() Tangent value
- tanh() Return the hyperbolic tangent
- xor() Perform bitwise XOR operation on integers

4 String Function

- *A*.string() Join all the members of a sequence with a delimiter
- asc() Obtain the Unicode value of the character at the specified position, if it is ASCII character, then return its ASCII code
- char() According to the given Unicode or ASCII code, get the corresponding characters
- fill() Obtain a string by filling characters in it
- isalpha() Judge if a string is composed of letters
- isdigit() Judge if a string is composed of numbers
- islower() Judge if a string is composed of letters in lower case
- isupper() Judge if a string is composed of letters in upper case
- left() Get the substring from the left of a string
- len() Compute the length of string
- like() Judge if a string matches the format string
- lower() Convert all characters to lower case
- mid() Return the substring of a string
- pad() Pad another character string before the string
- parse() Parse a string into the corresponding data type
- pos() Search the position of a substring in a parent string, and return null if not found
- rands() Get the random character string
- replace() Change the substring of a source string
- right() Get the substring from the right of a string

- s() Concatenate parameters into a string
- *s*.array() Split a string by delimiter so as to form a new sequence
- *s*.regex() Match the string member with the regular expression
- s.words() Select the English words out of a string
- string() Convert the object to the character type. Formatting is allowed during the process of conversion
- trim() Remove the space on both ends of a string
- upper() Convert all characters to upper case

5 Date Time Function

- after() Compute the new date which is certain days after a date
- age() Compute the number of whole years between a date and the current time
- date() Convert a string or integer to date
- date(*datetimeExp*) Get the date part of the datetime value
- datetime() Convert the string or long integer to date time
- datetime(*datetimeExp*) Adjust the precision of datetime formula and then return
- day() Get the day from a date
- days() Get the number of days of the year, quarter or month to which the specified date belongs
- deq() Judge if two dates are the same
- hour() Get the hour from a specified time
- interval() Compute the interval between two date time data
- millisecond() Get the millisecond from a time
- minute() Get the minute from a time
- month() Get the month from a date
- now() Get the current system date time
- pdate() Get the first and the last days of the week/month/quarter to which a date belongs
- periods() Generate a new sequence composed of datetimes
- second() Get the second from a time
- time() Convert the string or integer to time data
- time(*datetimeExp*) Get the time part of the datetime value
- workday(*t,k,h*) Compute a date time of n workdays from the specified date
- year() Get the year from a date

6 Sequence Conception & Member Accessing

- *A*(*i*) Get members from a sequence
- *A*(*i*)=*x* Assign value to members of a sequence
- *A*(*p*) Get members from a sequence according to an *n* integer sequence, so as to create a new sequence
- *A*(*p*)=*X* Correspondingly assign the data of ISeq *X* to the members of ISeq *p*.

ISeq *p* is composed of the sequence numbers of the members of the sequence A

- A(p)=x — Assign *x* to all members of the ISeq *p*. ISeq *p* is composed of the sequence numbers of the members of sequence A
- A.dup() — Copy a sequence
- A.len() — Get the length of a sequence
- A.m() — Get members at specified positions
- A.p() — Get sequence numbers of the members at the specified positions
- A.step() — Get members from a sequence with a starting position and a step, so as to create a new sequence
- A.to() — Get members from a sequence start from a specified position, so as to create a new sequence
- [] — A sequence having no member
- [a_1,…,a_n] — Define a sequence
- ifa() — To judge if an object is a sequence
- to() — Generate an integer sequence

7 Converge & Loop Function

- A.() — Compute an expression against each member of a sequence
- A.avg() — Compute the average value of all the non-null members in a sequence
- A.avg(x) — Compute x with each member of the sequence and then compute the average value of the non-null members of the new sequence
- A.calc() — Compute an expression against a specified record and return the result
- A.conj() — Concatenate all the members in a sequence whose members may also be sequence
- A.conj(x) — Compute x with each member of the sequence whose members are sequences, and then concatenate the computed results
- A.count() — Count the number of non-null members in a sequence
- A.count(x) — Compute x with each member of the sequence and then count the number of non-null sequence members of the new sequence
- A.diff() — Execute difference operation between the first member and the other members of a sequence
- A.diff(x) — Compute x with each member of the sequence whose members are sequences, and then perform difference operation between members of the new sequence
- A.ifn() — Get the first non-null member in a sequence
- A.ifn(x) — Compute x with each member of the sequence and return the first non-null member of the new sequence
- A.inv() — Compute the inverse Sequence
- A.isect() — Compute the intersection of all the member sequences of a sequence
- A.isect(x) — Compute x with each member of the sequence whose members are sequences, and then perform intersection operation between members

of the new sequence

- A.loop(x;a;c) Iterative loop of RSeq
- A.loops(x;a;c) Perform the cyclic iteration over RSeq and return the result of the last running of *x*
- A.max() Compute the maximum value of all the non-null members in a sequence
- A.max(x) Compute x with each member of the sequence and then compute the maximum value of the members of the new sequence
- A.merge() The merge operation will merge all sorted members and keep them in order
- A.min() Compute the minimum value of all the non-null members in a sequence
- A.min(x) Compute x with each member of the sequence and then compute the minimum value of the members of the new sequence
- A.rank() Compute the ranking of each member in a sequence
- A.rank(x) Get the ranking of sequence *A.*(*x*)
- A.ranki(y) Compute the ranking of a value in a sequence
- A.ranki(y,x) Get the ranking of a certain sequence member after the sequence is computed
- A.run() Compute expressions against each member in a sequence and return the sequence itself
- A.sum() Compute the sum of all the members in a sequence
- A.sum(x) Compute x with each member of the sequence and compute the summary value of the members of the new sequence
- A.union() Merge all the members in a sequence whose members may also be sequence
- A.union(x) Compute x with each member of the sequence whose members are sequences, and then perform union operation on members of the new sequence
- A.variance() Compute the variance value of all the non-null members in a sequence
- A.variance(x) Compute x with each member of the sequence and then compute the variance value of the members of the new sequence
- n.f(x) Compute a loop function using an integer as the loop variable
- p.inv() To compute the inverse ISeq of an ISeq

8 Locate & Pickup Function

- A.avgif(A_i:x_i,…) Locate the members in a sequence, and get the average of the members
- A.countif(A_i:x_i,…) Locate the members in a sequence, and count the members
- A.in(B) Judge if a sequence contains another sequence
- A.lookup() Locate all the positions of a member in a sequence, and get the members in these positions of another sequence

- A.maxif(A_i:x_i,…) Locate the members in a sequence, and get the maximum of the members
- A.maxp() Pick out the maximum member of a sequence
- A.minif(A_i:x_i,…) Locate the members in a sequence, and get the minimum of the members
- A.minp() Pick out the minimum member of a sequence
- A.pmax() Get the position of the maximum member of a sequence
- A.pmin() Get the position of the minimum member of a sequence
- A.pos() Get the position of a member in a sequence
- A.pos(x) Get the positions of some members of a sequence
- A.pseg(x) Return the position of a member in a sequence
- A.pselect() Get the positions of the selected members from a sequence
- A.psort() Get the positions of the sorted members of a sequence
- A.ptop() Get sequence numbers of top n smallest member of the sequence
- A.rvs() Generate a new sequence by reversing the members in a sequence
- A.select() Pick out members from a sequence which satisfied a condition
- A.sort() Generate a new sequence by sorting the members of a sequence
- A.sumif(A_i:x_i,…) Locate the members in a sequence, and get the sum of the members
- A.swap() Generate a new sequence by swapping the member positions of two specified intervals of a sequence
- A.top() Get the top n smallest records of the sequence member
- A.topx() Get the top n smallest values of a sequence

9 Calculation formula

- # In the range of the parents of the current cell, get the sequence number of the current cell among its peer cells
- ## In the range of parents of the current cell, get the number of peer cells of the current cell
- ==x Define a linked calculation expression of which the cell is called the linked calculation cell
- =x Define an instant calculation expression. This cell is the instant calculation cell
- @ Indicate the value of this cell if in any expression
- A[*L*] Get the A at the level *L* of the current cell
- C.~*c* Get the value of column C which shares the same row with the cell
- L# At the level L, get the sequence number of the current cell among its peer cells
- L## In the range of the level L, get the number of peer cells of the current cell
- L[*A*;*x*] In the range of cell L, the peer cells of A are clustered. Locate the peer cells that are relatively x away from the peer cells of the current A
- L{*A*;*a*:*b*} In the homocell set of A within the work scope of cell L, relative to

	the homocells of the current A, move a distance as specified by *a* till reaching the b. Then, return the homocell set of this band
➢ L{*A*}	In the range of cell L, get the cell set of peer cells of A
➢ [*a*:*b*]	Introduce the representation of cell sequence
➢ cr	Introduce the naming rule of cell
➢ num(*A*,*L*)	Get number of peer cells of A in the range of level L
➢ ord(*A*,*L*)	Get the sequence number of A among its peer cells in the range of level L
➢ pgall()	Get the total number of pages
➢ pgcell(*C*)	Get a sequence composed of values of all peer cells of C on the current page
➢ pgno()	Get the page number of the current page
➢ row()	Get the row number of the current row
➢ x	Define a constant
➢ {*a*:*b*}	Get a sequence composed of values of peer cells of a from a to b
➢ ~cr	Return the cell object, instead of the cell value

10 Finance

➢ Faccrint()	Caculate the accrued interest for a security that pays periodic interest
➢ Faccrintm()	Caculate the accrued interest for a security that pays interest at maturity
➢ Fcoupcd()	Caculate the coupon date
➢ Fcoups()	Caculate the number of coupons payable between a security's settlement date and maturity date/ the number of days in a coupon period that contains the settlement date/ the number of days from the beginning of a coupon's period to the settlement date / the number of days from the settlement date to the next coupon date
➢ Fdb()	Calculate the depreciation of an asset for a specified period
➢ Fddb()	Calculate the depreciation of an asset for a specified period
➢ Fdisc()	Calculates the discount rate for a security
➢ Fduration()	Return the modified duration of a security that pays periodic interest with an assumed par value
➢ Fintrate()	Calculate the interest rate for a security that pays interest at maturity
➢ Firr()	Calculate the internal rate of return for a series of cash flows represented by numeric values
➢ Fmirr()	Calculate the modified internal rate of return for a series of periodic cash flows
➢ Fnper()	Calculate the number of periods required to pay off a loan according to a specified periodic payment
➢ Fnpv()	Calculate the net present value of an investment

- Fpmt() Calculate each period's amount required to pay off an investment loan
- Fprice() Calculate the price of a security
- Frate() Calculate the interest rate required to pay an investment
- Freceived() Calculate the amount received at maturity for a security
- Fsln() Calculate the straight-line depreciation of an asset for each period
- Fsyd() Caculate depreciation of an asset for a specified period, using the sum-of-years' digits method
- Fv() Calculate the future value of an investment
- Fvdb() Calculate the depreciation of an asset for a specfied period
- Fyield() Caculate the yield rate

Function

#

Description:

In the range of the parents of the current cell, get the sequence number of the current cell among its peer cells

Syntax:

#

Remark:

Equals to **ord**(*current cell*). In the range of the parents of the current cell, get the sequence number of the current cell among its peer cells.

Parameters:

None

Return value:

In the range of parents of the current cell, the sequence number of current cell among its peer cells.

Example:

0	1	2		A	B	C	D	E	F
1-			1	Dept	ID	Name	Birthday	Salary	
	1-		2	Admin					=#
		1	3	Admin	1	Mike	1968-12-08	8000	=#
		1	4	Admin	4	Andy	1968-09-19	6000	=#
	2		5						
	1-		6	R&D					=#
		1	7	R&D	2	Jake	1962-02-19	9000	=#
		1	8	R&D	3	Lucy	1973-08-30	10000	=#
		1	9	R&D	5	Jim	1965-03-04	4000	=#
	2		10						

F2,F6 result: 1,2, equivalent to **ord(F2)** and **ord(F6)** respectively

F3-F4,F7-F9 result: 1,2,1,2, and 3. Take F3 for example, # equals to ord(F3)

##

Description:

In the range of parents of the current cell, get the number of peer cells of the current cell

Syntax:

##

Remark:

In the range of parents of the current cell, get the number of peer cells of the current cell, which is

equivalent to the **num**(*current cell*)

Parameters:

None

Return value:

In the range of the parent cell of the current cell, the number of peer cells of the current cell

Example:

0	1	2		A	B	C	D	E	H
1-			1	Dept	ID	Name	Birthday	Salary	
	1-		2	Admin					=##
		1	3	Admin	1	Mike	1968-12-08	8000	=##
		1	4	Admin	4	Andy	1968-09-19	6000	=##
	2		5						
	1-		6	R&D					=##
		1	7	R&D	2	Jake	1962-02-19	9000	=##
		1	8	R&D	3	Lucy	1973-08-30	10000	=##
		1	9	R&D	5	Jim	1965-03-04	4000	=##
	2		10						

H2,H6 result: 2, 2, which is equivalent to num(H2) and num(H6)

H3,H7 result: 2, 3. Take H3 for example, ## equals to num(H3)

${ *macroExp* **}**

Description:

This is used to complete the macro replacement operation.

Syntax:

${ *macroExp***}**

Remark:

The *macroExp* here is taken as an expression to compute, the calculation must be a string, and then the result will replace the **${** *macroExp***}**.

The macro enclosed in quotation marks or populated in the constant cell will not be replaced.

Parameters:

macroExp The macro expression, the calculation of which will replace the **${** *macroExp***}**, so the result must be a string.

Return value:

String.

Example:

0	1		A	B	C	D	E
1-		1	Student	PE	Math	English	History
	1	2	Aaron	="87"	="80"	="98"	="80"
2		3	==${B2}+3	=="${C2}+3"			

A3 result: 90. After replacing, the expression becomes **=87+3**, so the return value is **90**.

B3 result: "${C2}+3". Marco is enclosed in quotation marks, so it will not be replaced, and the return value is still "**${C2}+3**".

@

Description:

In the expression, it indicates the value of the current cell

Syntax:

@

Remark:

In the expression, it indicates the value of this cell

Parameter:

@ cell value

Return value:

Cell value

{a:b}

Description:

Get a sequence composed of values of peer cells of a from *a* to *b*

Syntax:

{a:b}

Remark:

Get a sequence composed of values of peer cells of a from *a* to *b*. Return null if out of the range.

If omitting *b*, then get a sequence composed of values of cells that are peers of a *and* in the same section as *a*, at the same level and in the same column to *a*.

Parameters:

a Starting cell

b Ending cell

Return value:

Sequence

Example:

0	1	2		A	B	C	D	E
1-			1	Dept	ID	Name	Birthday	Salary
	1-		2	Admin				
		1	3	Admin	1	Mike	1968-12-08	8000
		1	4	Admin	4	Andy	1968-09-19	6000
	2		5					
	1-		6	R&D				
		1	7	R&D	2	Jake	1962-02-19	9000
		1	8	R&D	3	Lucy	1973-08-30	10000

		1	9	R&D	5	Jim	1965-03-04	4000
	2		10					
2			11	={B3:B8}	={B3}			

A11 result: **[1,4,2,3]**

B11 result: **[1,4,2,3,5]**

~cr

Description:

Return the object of a cell, instead of the value of a cell

Syntax:

~cr

Remark:

Return the object of a cell, instead of the value of a cell

Parameter:

cr cell

Return value:

Object of a cell object

Example:

0	1		A	B
1-		1	9	4
	1	2	3	7
2		3	=~A2	

A3 result: A2, instead of 3

[a:b]

Description:

Introduce the representation of cell sequence.

Syntax:

[*a:b***]**

[*a:b,c,d:e***]**

Remark:

A sequence whose members are cell values from cell *a* to cell *b* in a same program cellset

Return value:

A sequence

Example:

0	1		A	B	C
1-		1			
	1	2	1	2	3
	2	3	4	5	6

2		4	=[A2:C3]		

A4 result: [1,2,3,4,5,6]. Return the sequence composed of all cell values in a rectangle range taking A2 and C3 as its diagonal points**.**

A()

A(i)

Description:

Get a member from a sequence.

Syntax:

A(*i*)

Remark:

Get the i^{th} member from the sequence *A*.

Parameters:

A sequence object

i sequence number expression of the member, which starts from 1.

Return value:

The member value of a specified sequence number in the sequence A

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
2		3	==[B2:F2](3)					

The value of **A3** is: **98**

Related concepts:

A (p)

A(p)

Description:

Get members from a sequence according to an *n* integer sequence, so as to create a new sequence.

Syntax:

A(*p*)

Remark:

A is an *n* sequence, and *p* is an *n* integer sequence whose length is *m*. Get the members in *p* in turn, using the member values of which as the sequence numbers to get the members in *A* to generate a new sequence.

Parameters:

p the *n* integer sequence whose length is *m* and member values are larger than 0 and less than or equal to the length of *A*. For example, if the length of sequence *A* is 5, then the members in *P* must be integers larger than 0 and less than or equal to 5. If *P* is an empty sequence, then return an empty sequence.

A a sequence whose length is *n*

Return value:

A new sequence whose length is *m*

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	91
	1	4	David	75	92	89	96	84
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8	==[B2:B7].sort()(to(3))	==[C2:C7]([1,3,35])	==[D2:D7]([])	==[E2:E7]([1,3,3])		
3		9						

The value of **A8** is: **[65,75,75]**

The value of **B8** is: error, Index is outside defined range

The value of **C8** is: **[]**

The value of **D8** is: **[80,96,96]**

Related concepts:

A (i)

A.( )

Description:

Compute an expression against each member of a sequence.

Syntax:

A.(*x*)

A.() return *A* itself

Remark:

Generate a new sequence composed of the results of the expression *x* against each member in *A*. "**~**" in *x* is used to reference the current member in *A*.

Parameters:

A a sequence

x an expression, "**~**" in which is used to reference the current member.

Return value:

The new sequence composed of the calculations of the expression *x* against each member in *A*

Example:

0	1		A	B	C	D	E	F
1-		1	EmployeeID	Dept	Name	Gender	Birthday	EntryDate
	1	2	1	Admin	Mike	Male	1968-12-08	1998-05-01
	1	3	2	R&D	Jake	Male	1962-02-19	2002-08-14
	1	4	3	Sales	Lucy	Female	1973-08-30	1999-04-01
	1	5	4	Admin	Andy	Male	1968-09-19	2000-05-03
	1	6	5	R&D	Jim	Male	1965-03-04	2005-10-17
2		7		==[E2]. (age(~))		=[A2].()		

B7 result: [43,49,38,43,46]

D7 result: [1,2,3,4,5]

A[L]

Description:

Get the A at the level *L* of the current cell

Syntax:

A[L]

Remark:

Get *A* in the range of the current cell at the level *L*. Without *[L]*, it is to get the *A* across the whole cellset.

Parameters:

L Cell; Only represent the layer

A Cell

Return value

Cell value

Example:

0	1	2		A	B	C	D	E
1-			1	Quarter	Month	Sales	Month-on-Month Comparison in this quarter	Month-on-Month Comparison in this Year
	1-		2	Quarter 1				
		1	3	Quarter 1	1	6000	==(C3-#REF![A2])/#REF![A2]	==(C3-#REF!)/#REF!
		1	4	Quarter 1	2	6800	==(C4-C3[A2])/C3[A2]	==(C4-C3)/C3
		1	5	Quarter 1	3	7500	==(C5-C4[A2])/C4[A2]	==(C5-C4)/C4
	1-		6	Quarter 2				
		1	7	Quarter 2	4	7200	==(C7-C5[A6])/C5[A6]	==(C7-C5)/C5
		1	8	Quarter 2	5	8100	==(C8-C7[A6])/C7[A6]	==(C8-C7)/C7
		1	9	Quarter 2	6	8000	==(C9-C8[A6])/C8[A6]	==(C9-C8)/C8

D3-D5,D7-D9 results are null, 0.13,0.10, null, 0.12, and -0.01, respectively

Of which, there is no previous month for D3. Therefore, the esCalc will automatically treat it as **#REF!**, and return the null result.

Of which, although there is a previous month for D7, the March is not belong to the second quarter, that

is, C5[A6] will return null. Therefore, the final result is null.

E3-E5,E7-E9 results are null, 0.13,0.10, -0.04, 0.12, and -0.01, respectively

Of which, since the previous month of E7 is March in the whole year, the -0.04 will be returned.

abs()

Description:

Compute the absolute value.

Syntax:

abs(*numberExp***)**

Remark:

Compute the absolute value of *numberExp*.

Parameters:

numberExp Data for which you want to compute the absolute value

Return value:

Numeric

Example:

- **abs(-3245.54)** **3245.54**
- **abs(-987)** **987**

acos()

Description:

Compute the arc cosine value

Syntax:

acos(*number***)**

Remark:

The parameter *number* is real number from -1 to 1

Parameters:

number The real number for which you want to compute the arc cosine

Return value:

Arc consine

Example:

- **acos(-1)** **3.141592653589793**
- **acos(cos(pi()/2))** **1.5707963267948966**
- **acos(cos(0))** **0.0**

Related concepts:

asin()

atan()

acosh()

Description:

Return the inverse hyperbolic cosine

Syntax:

acosh(*number***)**

Remark:

The parameter *number* is a real number

Parameter:

number The real number for which you want to find the inverse hyperbolic cosine

Return value:

The inverse hyperbolic cosine

Example:

acosh(10) **2.993222846126381**

after()

Description:

Compute the new date which is certain days after a date

Syntax:

after (*dateExp, n***)**

dateExp ± *n* **after (***dateExp, n***)**

Remark:

Compute the new date which is *n* days/*n* months/*n* years after the date *dateExp*

If the day does not exist, then return the last day of the month or the year.

For example, **after@m("2009-03-31",-1)** returns **2009-02-28**

Parameters:

dateExp The starting date expression whose result is a date, time or date time,

n The integer expression of which the positive integer indicates to compute a new date of *n* days/years/months later, and the negative integer indicates to compute a new date of *n* days/years/months before

Options:

@y Compute the new date which is *n* years after the specified date

@q Compute the new date which is n quarters since the specified date

@m Compute the new date which is *n* months after the specified date

@e If the specified date is the last day of a month, the computed new date will not be adjusted to the last day of the month to which it belongs. By default the new date will be adjusted to the last day of the corresponding month. This option works with @yqm.

@s Compute the date time which is *n* seconds after the specified date

@ms Compute the date time which is *n* milliseconds after the specified date

The default is to compute the new date which is *n* days after the specified date

Return value:

Date time

Example:

- **after(datetime("19800227","yyyyMMdd"),5)** **1980-03-03 00:00:00**
- **after@y(datetime("19800227","yyyyMMdd"),5)** **1985-02-27 00:00:00**
- **after@q(datetime("19800227","yyyyMMdd"),5)** **1981-05-27 00:00:00**
- **after@m(datetime("19800227","yyyyMMdd"),5)** **1980-07-27 00:00:00**
- **after@s(datetime("19800227","yyyyMMdd"),5)** **1980-02-27 00:00:05**
- **after@ms(datetime("19800227","yyyyMMdd"),5)** **1980-02-27 00:00:00**
- **after(datetime("19800227","yyyyMMdd"),-3)** **1980-02-24 00:00:00**
- **after@y(datetime("19800227","yyyyMMdd"),-3)** **1977-02-27 00:00:00**
- **after@q(datetime("19800227","yyyyMMdd"),-3)** **1979-05-27 00:00:00**
- **after@m(datetime("19800227","yyyyMMdd"),-3)** **1979-11-27 00:00:00**
- **after@s(datetime("19800227","yyyyMMdd"),-3)** **1980-02-26 23:59:57**
- **after@ms(datetime("19800227","yyyyMMdd"),-3)** **1980-02-26 23:59:59**
- **datetime("19800227","yyyyMMdd")+5** **1980-03-03 00:00:00**
- **datetime("19800227","yyyyMMdd")-5** **1980-02-22 00:00:00**
- **after@ey(datetime("19770228","yyyyMMdd"),3)** **1980-02-28 00:00:00**
- **after@eq(datetime("19800229","yyyyMMdd"),1)** **1980-05-29 00:00:00**
- **after@em(datetime("19800229","yyyyMMdd"),5)** **1980-07-29 00:00:00**

age()

Description:

Compute the number of whole years between a date and the current time

Syntax:

age(*dateExp***{,** *formatExp* **})**

age(*stringExp***,***formatExp***)**

Remark:

Compute the number of whole years between the date *dateExp* and the current time

Parameters:

dateExp Date expression whose result is the date

stringExp String expression whose result is normal date or Chinese datetime format string

formatExp Format the expression, such as "**yyyyMMdd**","**yyyy-MM-dd**"

Options:

@y	The calculation is corrected to the year
@m	The calculation is corrected to the month
	The calculation is corrected to the day by default

Return value:

Integer

Example:

- **age(date ("1980-09-01"))**
- **age@m(datetime("1980-09-01 12:23:56"))**
- **age@y("19800227","yyyyMMdd")**

Alignment Arithmetic Operation

Description:

Generate a new sequence by Alignment Arithmetic Operation between two sequences which are of the same length, such as aligning add, aligning subtract, aligning multiply and so on.

Syntax:

A++*B*	**[***A***(1)+***B***(1),***A***(2)+***B***(2),...],** Aligning add
A--*B*	**[***A***(1)-***B***(1),***A***(2)-***B***(2),...],** Aligning subtract
*A****B*	**[***A***(1)****B***(1),***A***(2)****B***(2),...],** Aligning multiply
A//*B*	**[***A***(1)/***B***(1),***A***(2)/***B***(2),...],** Aligning divide
A%%*B*	**[***A***(1)%***B***(1),***A***(2)%***B***(2),...],** Aligning division and get the remainder
*A**B*	**[***A***(1)***B***(1),***A***(2)***B***(2),...],** Aligning division and get the integer value

Remark:

Alignment Arithmetic Operation means calculate the two members in the same position of *A* and *B* one by one, and thus generate new members for the new sequence. For example, *A*++*B* indicates **[***A***(1)+***B***(1),***A***(2)+***B***(2),...].**

Parameters:

A	an *n* sequence
B	an *n* sequence

Return value:

A new sequence after the aligning computation

Example:

0	1		A	
1-		1	=[4,2,3,3]++[5,10,2,1]	[9,12,5,4]
	1	2	=[4,2,3,3]--[5,10,2,1]	[-1,-8,1,2]
2		3	=[4,2,3,3]**[5,10,2,1]	[20,20,6,3]
3		4	=[4,2,3,3]//[5,10,2,1]	[0.8,0,2,1.5,3.0]
4		5	=[7,12,3,3]%%[5,10,2,1]	[2,2,1,0]
5		6	=[7,12,3,3]\\[5,10,2,1]	[1,1,1,3]

Related concepts:

Difference sequence

Intersection sequence

and()

Description:

Perform bitwise operation on integers

Syntax:

and($x_{i,...}$**)**
and(*A***)**

Remark:

Perform bitwise operation on integers

Parameter:

A	Sequence
x_i	The numerical expression based on which you perform the bitwise AND operation

Return value:

An integer

Example:

and(6,11)	**2**

Arithmetic Operation

Description:

Perform the four arithmetic operations on two members.

Syntax:

*x***+***y*
*x***-***y*
*x******y*
x/*y*

Remark:

If both the first-progression computation (addition and subtraction) and the second-progression computation (multiplication and division) appear in the same formula, then the operational order will

be first the multiplication and division, and then the addition and subtraction.

In case there is any bracket, firstly work out the factors inside the bracket and then those outside.

For the same progression, work it out from left to right.

Parameters:

x Numeric

y Numeric

Return value:

Numeric

Example:

0	1		A	
1-		1	=2+5	7
	1	2	=2-5	-3
2		3	=2*5	10
3		4	=10/5	2.0
4		5	=2+3.5*30	107.0

array()

s.array()

Description:

Split a string by delimiter so as to form a new sequence.

Syntax:

s.**array(***d***)**

Remark:

Split string *s* by delimiter *d* and form a new sequence. The data type of the members of the new sequence will be processed by default, that is, consider the number characters as digital values, **[]** as a sequence, **2001-01-01** as a date, and so on.

Options:

@s Split into a sequence of strings, the data type will not be processed. Those delimiters between the quotation marks or the brackets will be ignored.

@1 This option is a sub option of **@s**. Stop searching and split string into 2 parts by the first *d* found

@b This option is a sub option of **@s**. Those delimiters between the quotation marks or the brackets will not be ignored and the data type will be processed

Parameters:

s the string to be spitted

d the delimiter; If it is omitted, comma is the default

Return value:

The new sequence generated by splitting the string *s*

Example:

0	1		A	B	C	D	E

1-		1	==”1,[a,b],(2,c),'5,6”'				
	1	2		==A1.array()	==A1.array@1()	=A1.array@s()	=A1.array@b()
2		3	==”a:b:c”				
3		4		==A3.array(":")			

B2 result: **[1,["a","b"],"(2,c)","5,6"]**

C2 result: **[1,"[a,b],(2,c),\'5,6\'"]**

D2 result: **["1","[a,b]","(2,c)","\'5,6\'"]**

E2 result: **[1,"[a","b]","(2","c)","\'5","6\'"]**

B4 result:**["a","b","c"]**

Related concepts:

A.string()

asc()

Description:

To obtain the Unicode value of the character at the specified position, if it is ASCII character, then return its ASCII code.

Syntax:

asc(*string*{, *nPos*} **)**

Remark:

To obtain the Unicode value of the character at the specified position *nPos* of *string*, if it is ASCII character, then return its ASCII code.

In general, the English character and its extended character are all the ASCII character; Chinese, Japanese, Korean, and other Asian characters are all the Unicode character. ASCII character is an 8 bit character set, and the Unicode character is a 16 bit character set, of which 3 bits are used to indicate the character type.

Parameters:

string The given strings

nPos Integer expression, the default is **1**

Return value:

Integer

Example:

- **asc("def")** **100 (ascii)**
- **asc("def",2)** **101 (ascii)**
- **asc("China")** **67 (unicode)**
- **asc("China",2)** **104(unicode)**

Related concepts:

char()

asin()

Description:

Compute the arc sine value.

Syntax:

asin(*number***)**

Remark:

The parameter *number* is real number from -1 to 1

Parameters:

number The real number for which you want to compute the arcsine value

Return value:

Arcsine value

Example:

- **asin(-1)** **-1.5707963267948966**
- **asin(sin(pi()/2))** **1.5707963267948966**
- **asin(sin(0))** **0.0**

Related concepts:

acos()

atan()

asinh()

Description:

Return the inverse hyperbolic sine

Syntax:

asinh(*number***)**

Remark:

The parameter *number* is a real number

Parameter:

number The real number for which you want to find the inverse hyperbolic sine

Return value:

The inverse hyperbolic sine

Example:

asinh(10) **2.99822295029797**

Assignment

Description:

Assign value to a member of a sequence.

Syntax:

A(*i*)=*x* Assign the i^{th} member of the sequence *A* with the value *x*. If *i* exceeds the number of members of *A*, then an error will be raised.

A(*p*)=*x* Assign all members of the ISeq *p* with the value *x*. If the number of members of *p* exceeds the number of members of *A*, then an error will be raised.

A(*p*)=X Correspondingly assign the members of ISeq *p* with the data from ISeq *X*. The length of these two integer sequences must be the same. Otherwise, an error will be raised.

Parameters:

A Sequence

i Sequence number of the member in the sequence

x Data you are about to assign to the member

p ISeq composed of the sequence numbers

X ISeq for assigning values to members

Example:

0	1		A	
1-		1	[12,23,34]	
	1	2	=A1(2)=11	A1 result: [12,11,34]
2		3	=A1([1,2,3])=6	A1 result: [6,6,6]
3		4	=A1([1,2,3])=[9,8,7]	A1 result: [9,8,7]

Related concepts:

atan()

Description:

Compute the arc tangent value.

Syntax:

atan (*number***)**

Remark:

The parameter *number* is real number

Parameters:

number Real number for which you want to compute the arctangent

Return value:

Arc tangent

Example:

- **atan(1)** **0.7853981633974483**
- **atan(tan(pi()/2))** **1.5707963267948966**
- **atan(tan(0))** **0.0**

Related concepts:

asin()

acos()

atanh()

Description:

Return the inverse hyperbolic tangent

Syntax:

atanh(*number***)**

Remark:

The parameter is a real number

Parameter:

number Any real number between -1 and 1

Return value:

The inverse hyperbolic tangent

Example:

atanh(0.5) **0.5493061443340549**

avg()

A.avg()

Description:

Compute the average value of all the non-null members in a sequence.

Syntax:

A.**avg()** Equivalent to **avg**($x_1,\ldots,x_n$)

Remark:

Compute the average value of all the non-null members in the sequence *A*, which is equal to *A*.**sum()**/*A*.**count()**. If the number of all the non-null members is **0**, the average value is null. Ignore if the member is not numeric value.

Parameters:

A A sequence

Return value:

The average value of all the non-null members in the sequence *A*

Example:

0	1		A	B	C	D	E	F	G
1-		1	Student	PE	Math	English	History	Geography	AVG
	1	2	Aaron	87	80	98	80	98	==[B2:F2].avg()
	1	3	Charles	90	99	80	76	91	==[B3:F3].avg()
	1	4	David	75	92	89	96	84	==[B4:F4].avg()
	1	5	Mary	93	78	81	92	76	==[B5:F5].avg()
	1	6	Vincent	75	90	88	92	97	==[B6:F6].avg()
	1	7	Lucy	65	71	89	69	92	==[B7:F7].avg()
		8	Lily	aaa	71	89	69	92	==[B8:F8].avg()
2		9		==avg(87,null,75,93,75,65,50)					

G2-G8 results: 88.6,87.2,87.2,84.0,88.4,77.2, 80.25

B9 result: 74.16666666666667

Related concepts:

A.sum()

A.count()

A.min()

A.max()

A.variance()

A.avg(x)

A.avg(x)

Description:

Compute x on each member of the sequence and then compute the average value of the non-null sequence members.

Syntax:

A.**avg(***x***)** Equivalent to *A*.(*x*).**avg**()

Remark:

Compute x on each member of the sequence and return the average value of the non-null members of the new sequence

Parameters:

A A sequence

x an expression, "**~**" in which is used to reference the current member.

Return value:

Numerical values

Example:

0	1		A	B	C	D	E	F	G	H
1-		1	Student	PE	Math	English	History	Geography	AVG	AVG

	1	2	Aaron	87	80	98	80	98	==[B2:F2].avg(~)	==[B2:F2].(~+10).avg()
	1	3	Charles	90	99	80	76	91	==[B3:F3].avg(~)	==[B3:F3].(~+10).avg()
	1	4	David	75	92	89	96	84	==[B4:F4].avg(~)	==[B4:F4].(~+10).avg()
	1	5	Mary	93	78	81	92	76	==[B5:F5].avg(~)	==[B5:F5].(~+10).avg()
	1	6	Vincent	75	90	88	92	97	==[B6:F6].avg(~)	==[B6:F6].(~+10).avg()
	1	7	Lucy	65	71	89	69	92	==[B7:F7].avg(~)	==[B7:F7].(~+10).avg()
	1	8	Lily	aaa	71	89	69	92	==[B8:F8].avg(~)	==[B8:F8].(~+10).avg()
2		9								

G2-G8 results: 88.6,87.2,87.2,84.0,88.4,77.2, 80.25

H2-H8 results: 98.6,97.2,97.2,94.0,98.4,87.2,74.2

Related concepts:

A.avg()

avgif()

A.avgif()

Description:

Locate all the positions of a member in a sequence, and get the average of the members in these positions of another sequence.

Syntax:

A.**avgif(***A*$_i$:*x*$_i$, ...**)**

Remark:

Locate all the positions of member x_i in A_i, acquiring the intersection of these positions and return the average value of the non-null members in these positions of *A*

Parameters:

A_i a sequence

x_i the members in A_i

A the target sequence

Return value:

The average value of the non-null members in those result positions of *A*

Example:

0	1		A	B	C	D
1-		1	Class	Name	Subject	Score
	1	2	class one	Aaron	PE	80
	1	3	class one	Bill	PE	89
	1	4	class one	Chris	Math	98
	1	5	class two	Jack	PE	78
	1	6	class two	Chris	PE	90
	1	7	class two	Jack	Math	93
	1	8	class two	Aaron	Math	85
	1	9	class one	Bill	Math	89

2		10	={D2}.avgif({C2}:"PE")
3		11	={D2}.avgif({C2}:"PE",{A2}:"class one")

A10 result: 84.25

A11 result: 84.5

Related concepts:

A.countif(A_i:x_i,…)

A.sumif(A_i:x_i,…)

A.minif(A_i:x_i,…)

A.maxif(A_i:x_i,…)

Batch computation

Description:

To compute a series of expressions one by one in an automated fashion, and return the result of the last expression.

Syntax:

(x_1,x_2,…,x_k)

Remark:

The later expressions can refer to the variable value calculated and assigned by the preceding expressions.

Parameters:

x_k Expressions you want to compute in a batch

Example:

- **(1,1+2,2+3)** **5**
- **(a=1,b=a*3,c=b+5,a+b+c)** **12**

bits()

Description:

Convert to a decimal number

Syntax:

bits($x_{i,...}$**)**

Remark:

This function equals sum($2^{i-1}*x_i$), which converts a number of other numeral systems to a decimal integer. First convert x_i to an integer if it is a string.

If there is only a single x_i and it is a string, split it into a sequence of a single character first.

Parameters:

x_i The integer/string to be converted

Options:

@h Equivalent to sum($16^{i-1}*x_i$). First convert x_i to the integer according to the rules of hexadecimal numeral system if it is the string

@d Equivalent to sum($10^{i-1}*x_i$). First convert x_i to the integer if it is the string

@n Equivalent to sum(2^{i-1}*if(x_i,1,0)). Convert null or nonnull x_i to corresponding value

@s It works with other options to return a string that is equivalent to the number of the specified numeral system

Return value:

Numeric value

Example:

- **=bits([1,0,1,1])** **13**
- **=bits("1011")** **13**, Split the single string into a sequence. It is equal to =bits("1","0","1","1")
- **=bits@d(1,1,1,5)** **5111**
- **=bits@n(1,1,1,5)** **15**
- **=bits@h("A",1,1,5)** **20762**
- **=bits@sn(12)** **1**
- **=bits@sd(12)** **12**
- **=bits@sh(1212)** **4bc**

C.~c

Description:

Get the value of column *c* that is in the same row of cell *C*

Syntax:

C.~c

Remark:

Get the value of column *c* that is in the same row of cell *C*

Parameter:

C cell

c column

Return value:

value

Example:

0	1		A	B
1-		1	9	4
	1	2	3	7

2		3	=~A2.~B	

The result of A3 is 7

calc()

A.calc()

Description:

Compute an expression against a specified member and return the result.

Syntax:

A.**calc**(*k*,*x*) compute *x* against the k[th] member of *A* and return the result

A.**calc**(*p*,*x*) compute *x* against the members of *A* specified by the integer sequence *p* and return the result sequence

Remark:

Compute an expression against a specified member and return the result.

Parameters:

A a sequence

x an expression, "**~**" is used to reference the current member.

k an integer, specifying which member it is

p an integer sequence, specifying which members they are

Return value:

A calculation of *x* or a sequence composed of the calculations of *x*

Example:

- Compute *x* against the k^{th} member

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	90	98	80	98
2		3	==[B2:F2].calc(3,~*2)					

A3 result: 196

Compute expression "**~*2**" against the third member of sequence **[B2:F2]**, "**~**" indicates the current member. The result is **196**.

- compute *x* against the members of *A* specified by the integer sequence *p*

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	90	98	80	98
2		3	==[B2:F2].calc([2,4],~*2)					

A3 result: [180,160]

Compute expression "**~*2**" against the forth and the third members of sequence **[B2:F2]**separately and the result is **[180,160]**.

case()

Description:

According to the various results of judge expressions, return various values.

Syntax:

case(x,x_1:y_1,…,x_k:y_k;y**)**

Remark:

This function will compute from left to the right. Compute the judge expression x first, and then compute the other x_k. If there is any result of expression x_k equals to the result of x, then return the result of y_k, and the calculation is terminated. If none of the result of expression x_k equals to the result of x, and there exists default expression y, then return the result of y, otherwise, return **null**.

Parameters:

x	Judge expression
x_k	Value expression
y_k	Result expression
y	Default expression.

Example:

0			A	B	
1-		1	DeptId	DeptName	
	1	2	1	==case(A2,1:"Dept 1",2:"Dept 2",3:"Dept 3";"Admin Dept")	The value of B2 is "Dept 1"
	1	3	4	==case(A3,1:"Dept 1",2:"Dept 2",3:"Dept 3";"Admin Dept")	The value of B3 is "Admin Dept"

Related concepts:

if()

in()

ceil()

Description:

Truncate the data at the specified position, and carry the remaining part if any.

Syntax:

ceil(*numberExp*, {*nExp*}**)**

Remark:

Truncate the data *numberExp* at the specified position *nExp*, and carry the remaining part (if any).

Parameters:

numberExp	Data to be truncated
nExp	Integer number for specifying the truncation position, >0: Move the decimal point to the right for *nExp* places, <0: Move the decimal point to the left for *nExp* places, =0: Indicate the current decimal places.

Return value:

Numeric

Example:

- **ceil(3450001.004,0)** **3450002.0**
- **ceil(3450001.004,-1)** **3450010.0**
- **ceil(3450001.004,-2)** **3450100.0**
- **ceil(3450001.004,1)** **3450001.1**
- **ceil(3450001.004,2)** **3450001.01**

Related concepts:

floor()

round()

char()

Description:

According to the given Unicode or ASCII code, get the corresponding characters.

Syntax:

char(*int***)**

Remark:

In general, the English character and its extended characters are all the ASCII code; Chinese, Japanese, Korean, and other Asian characters are all Unicode characters. ASCII character is an 8 bit character set, and Unicode character is a 16 bit character set, of which 3 bits are used to indicate the character type.

Parameters:

int Integer expression, Unicode code or ASCII code

Return value:

Character

Example:

- **char(87)** **'W'**

Related concepts:

asc()

cmp()

Description:

Compare the value of two expressions or two sequences

Syntax:

cmp(*x, y***)**

cmp(*A*{, *B*}**)**

cmp(*A*,**0)**

Remark:

Compare the value of two expressions *x* and *y* or two sequences *A* and *B*. An error will be reported if *x* and *y* or *A* and *B* cannot be compared*:*

When comparing the value of two expressions *x* and *y*, return **0** if they are equal; return **1** if *x* is greater

than *y*; return **-1** if *x* is less than *y*.

When comparing two sequences *A* and *B*, compare the two members in the same position of *A* and *B* one by one. Return **0** if all the members are equal; for the first members those are not equal, return **1** if the one in *A* is larger and return **-1** if the one in *A* is smaller. If the members in the sequence *A* and *B* are not the same, and their beginning members are the same, then the value with less members will be smaller.

Parameters:

x	expression
y	expression
A	an *n* sequence
B	an *m* sequence. If sequence *B* does not exist, then it will be taken as a **0** sequence by default, that is, comparison will be done between sequence *A* and sequence **[0…0]**.

Example:

0	1	2		A	B	C	D	E
1-			1	Class	Name	PE	Math	Sum
	1-		2	class one				
		1	3	class one	Bill	82	63	==C3+D3
		1	4	class one	Chris	99	63	==C4+D4
		1	5	class one	Jack	99	92	==C5+D5
	1-		6	class two				
		1	7	class two	Jack	71	63	==C7+D7
		1	8	class two	Aaron	82	63	==C8+D8
		1	9	class two	Chris	89	81	==C9+D9
2			10	==cmp(C3,C4)	==cmp(B3,B4)	==cmp([E3:E5],[E7:E9])	==cmp([E3:E5])	==cmp(C3,B3)

A10 result: -1

B10 result: -1

C10 result: 1

D10 result: 1. The operation is the same as **cmp([E3:E5],[0,0,0])**

E10 result: The numeric value and character string cannot be compared

Related concepts:

Difference sequence

Intersection sequence

Sequence Union

Alignment Arithmetic Operation

Concatenate sequence

combin()

Description:

Return the number of combinations

Syntax:

combin(*n*,*k*)

Remark:

The number of ways of picking *k* elements from *n* objects

Parameters:

n An integer that is the number of given objects

k An integer that is the number of elements you want to pick from the given set of objects

Return value:

A number

Example:

combin(8,2)	**28**

Comparison operation

Description:

The comparison operations between two numerical values, sequences, or characters

Syntax:

x==*y* If values of two operands are the same, then the result is true. Otherwise, it is false.

x! =*y* If values of two operands are not the same, then the result is true. Otherwise, it is false.

x<*y* If x is less than y, then the result is true. Otherwise, it is false.

x>*y* If x is greater than y, then the result is true. Otherwise, it is false.

x<=*y* If x is not greater than y, then the result is true. Otherwise, it is false.

x>=*y* If x is not less than y, then the result is true. Otherwise, it is false.

Remark:

The comparison operation requires the numeric or character operands, and the result is the boolean value. For those Not Less Than (or Not Greater Than) relational operators, the result value is false only when both the Greater and Equal relations are not true, and the result is still true as long as either of the relations is true.

If *x* and *y* are sequences, then *x* and *y* will be compared in one-to-one relation following the sequential order of members. Return the result according to the first unequal align comparison of members, return true if the conditions are met. Otherwise, return false. However, for the align comparison ==, the true will only be returned when all members are equal in the align comparison. Otherwise, return false. If the numbers of members in sequence *x* and *y* are not the same, and their beginning members are the same, then the value with less members will be smaller.

Parameter:

x Sequence, numerical, or character data

y Sequence, numerical, or character data

Return value:

true/false

Example:

0	1		A	
1-		1	=2==3	false
	1	2	=2!=5	true
2		3	=2>5	false
3		4	=10<5	false
4		5	=2<=3	true
5		6	=3>=4	false
6		7	="a"=="b"	false
7		8	=[5,2,1,2]<=[5,2,1,2,-3]	true

Compound assignment

Description:

The value of a certain variable will undergo a certain computation with an expression to assign to the variable itself.

Syntax:

a?=*x*

Remark:

A variable value will undergo a certain computation with an expression to assign to the variable itself.

Parameters:

a Variable name

x Legal expression

? Operators, support the operator +, -, *, /, \, and %

Return value:

New variable

Example:

0	1		A
1-		1	=a=7
	1	2	=a-=4

The value of A2 is:3

Concatenate sequence

Description:

Concatenate two sequences so as to generate a new sequence.

Syntax:

A|*B*

Remark:

Concatenate the members (or single values) of two sequences in proper order to compose a new sequence, that is, **[***A***(1),...,***A***(***n***),***B***(1)**,…,***B***(***m***)]**.

Parameters:

A an *n* sequence or a single value; When it is a single value, it is regarded as **[***A***]**

B an *m* sequence or a single value; When it is a single value, it is regarded as **[***B***]**

Return value:

The new sequence after concatenating the two sequences *A* and *B*

Example:

0	1		A	B
1-		1	Student	English
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	73
	1	8	Lily	69
	1	9	Peter	64
2		10	==\{A2\}	==A10(to(3))|A10(to(A10.len()-2,A10.len()))

List the top 3 and the last 3 students respectively

The value of **B10** is: **["Aaron","Charles","David","Lucy","Lily","Peter"]**

0	1		A	B
1-		1	Student	English
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	Math
	1	9	Vincent	100
	1	10	Aaron	99
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==\{A2\}(to(3))|\{A9\}(to(3))	

The student whose math or English is among the top 3

The value of **A15** is: **["Aaron","Charles","David","Vincent","Aaron","Charles"]**

The same member will appear repeatedly.

Related concepts:

Difference sequence
Intersection sequence
Sequence Union
Alignment Arithmetic Operation
cmp()

conj()

A.conj()

Description:

Concatenate all the members in a sequence whose members may also be sequence.

Syntax:

A.conj()

Remark:

Generate a new sequence by concatenating all the members in sequence A whose members may also be sequence.

Parameters:

A A sequence whose members are sequences

Return value:

The new sequence by concatenating all the members in sequence A

Example:

0	1	2		A	B	C
1-			1	Class	Name	Score
	1-		2	class one		
		1	3	class one	Bill	89
		1	4	class one	Chris	98
		1	5	class one	Jack	78
	1-		6	class two		
		1	7	class two	Jack	93
		1	8	class two	Aaron	85
		1	9	class two	Chris	89
2			10	==[{B3},{B7}].conj()	==[{B3},"Jack","Petter"].conj()	

A10 result: ["Bill","Chris","Jack","Jack","Aaron","Chris","Bill","Chris","Jack","Jack","Aaron","Chris"]

B10 result: ["Bill","Chris","Jack","Jack","Aaron","Chris","Jack","Petter"]

Related concepts:

A.union()
A.diff()
A.isect()

A.conj(x)

Description:

Compute x with each member of the sequence whose members are sequences, and then concatenate the computed results.

Synatax:

A.conj(x)

Remark:

Compute x on sequence *A*, whose members are sequences, by loop, and then concatenate the computed results to form a new sequence.

Parameters:

A A sequence whose members are sequences

x an expression, "**~**" in which is used to reference the current member.

Return value:

A sequence

Example:

0	1		A	B
1-		1	Student	Math
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	PE
	1	9	Vincent	100
	1	10	Aaron	99
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==[{B2},{B9}].conj(~-10)	

A15 result: [88,85,77,73,65,55,90,89,82,78,70,61]

Related concepts:

A.conj()

Constant Cell

Description:

Define a constant

Syntax:

x or *'x*

Remark:

Define a constant, and the cell value is the constant or string of *x*. If the string x is started with "=" or "==", then you can write it as *'x* to avoid misunderstanding it as instant calculation cell or linked calculation cell.

Parameters:

x Constant or string

Return value:

x

Example:

- **12345 The cell value is the numeric value 12345**
- **abc The cell value is the string abc**
- **'=1+1 The cell value is the string =1+1**

Constant sequence

Description:

Define a sequence.

Syntax:

[a_1… a_n**]**

Remark:

Define a sequence composed of n members a_1… a_n.

Parameters:

a_n Member of sequence.

Return value:

A sequence composed of a_1… a_n.

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
2		3	==[B2:F2]	==[B2,C2,D2]	==[]	==[B2*2,F2]		

The value of **A3** is: **[87,80,98,80,98]**

The value of **B3** is: **[87,80,98]**

The value of **C3** is: **[]**, An empty sequence

The value of **D3** is: **[174,98]** A sequence composed of **174** and **98**.

cos()

Description:

Compute the cosine value

Syntax:

cos(*numberExp***)**

Remark:

The parameter *numberExp* is in radians.

Parameters:

numberExp The radian number of the cosine to be calculated

Return value:

float type

Example:

- **cos(pi())** **-1.0**
- **cos(pi(2))** **1.0**

Related concepts:

sin()

tan()

cosh()

Description:

Return the hyperbolic cosine

Syntax:

cosh(*number***)**

Remark:

The parameter *number* is a real number

Parameter:

number The real number for which you want to find the hyperbolic cosine

Return value:

The hyperbolic cosine

Example:

cosh(4) **27.308232836016487**

count()

A.count()

Description:

Count the number of non-null members in a sequence.

Syntax:

A.**count()** Equivalent to **count**(x_1,…,x_n)

Remark:

Count the number of the non-null members in the sequence *A*.

Parameters:

A an *n* sequence

Return value:

The integer which is the number of the non-null members in the sequence *A*

Example:

0	1		A	B	C
1-		1	Student	PE	Math
	1	2	Aaron	87	80
	1	3	Charles	90	99
	1	4	David	75	92
	1	5	Mary	93	78
	1	6	Vincent		90
	1	7	Lucy	65	71
	1	8	Lily	50	89
2		9	=count(1,null,3,4)	==\{B2\}.count()	==\{C2\}.count()

B9,C9. Count the number of examinees

B9 result: 6

C9 result: 7

A9 result:3

Related concepts:

A.sum()

A.avg()

A.min()

A.max()

A.variance()

A.count(x)

A.count(x)

Description:

Count the number of non-null members in a sequence.

Syntax:

A.**count(***x***)** Equivalent to *A*.(*x*).**count**()

Remark:

Compute *x* by looping *A* and return number of the records that can make *x* being not null.

Parameters:

A A sequence

x an expression, "~" in which is used to reference the current member.

Return value:

An integer

Example:

0	1		A	B
1-		1	Student	Math
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	PE
	1	9	Vincent	100
	1	10	Aaron	99
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==\{B9\}.count(~-10)	==\{B9\}.(~-10).count()

A15,B15 results:6

Related concepts:

A.count()

countif()

A.countif()

Description:

Locate all the positions of a member in a sequence, and count the members in these positions of another sequence.

Syntax:

A.**countif(**A_i:x_i,…**)**

Remark:

Locate all the positions of member x_i in A_i, acquiring the intersection of these positions and return the

count of the non-null members in these positions of *A*

Parameters:

A_i a sequence

x_i the members in A_i

A the target sequence

Return value:

The count of the non-null members in those result positions of *A*

Example:

0	1		A	B	C	D
1-		1	Class	Name	Subject	Score
	1	2	class one	Aaron	PE	Excellent
	1	3	class one	Bill	PE	Good
	1	4	class one	Chris	Math	Excellent
	1	5	class two	Jack	PE	Excellent
	1	6	class two	Chris	PE	Good
	1	7	class two	Jack	Math	Excellent
	1	8	class two	Aaron	Math	Good
	1	9	class one	Bill	Math	Excellent
2		10	={B2}.countif({C2}:"PE")			
3		11	={B2}.countif({C2}:"PE",{D2}:"Excellent")			

A10 result: 4

A11 result: 2

Related concepts:

A.sumif(A_i:x_i,…)

A.avgif(A_i:x_i,…)

A.minif(A_i:x_i,…)

A.maxif(A_i:x_i,…)

cr

Description:

Introduce the naming rule of cell.

Syntax:

Cr or $*C*$*r*

Remark:

C is a column number starting from **A** and *r* indicates a row number starting from 1.

For example, **=A1** indicates the cell **A1**. The naming rule of cells here is the same as that of Microsoft Office Excel.

When pasting, *Cr* will adjust and paste automatically while $*C*$*r* will not.

Example:

0	1		A	B

1-		1	1	2
	1	2	=A1+1	
2		3	=$A$1	

If selecting **A2**, then paste it to **B2** cell. The **A1** in the expression **=A1+1** will auto adjust and paste to **B1**

If selecting **A3**, then paste it to **B3** cell. The **$A$1** in the expression **=$A$1** will not adjust automatically.

0	1		A	B
1-		1	1	2
	1	2	=A1+1	=B1+1
2		3	=$A$1	=$A$1

date()

date(datetimeExp)

Description:

Get the date part of the datetime value

Syntax:

date(*datetimeExp***)**

Remarks:

Get the date part of the *datetimeExp*

Parameters:

datetimeExp datetime

Return value:

Date

Example:

- **date(now())** **2013-12-09**

Related concepts:

date()

datetime(*datetimeExp*)

time(*datetimeExp*)

datetime()

time()

date()

Description:

Convert a string or integer to date

Syntax:

date(*stringExp*,*format***)** Convert the type of *stringExp* to date according to the format defined by *format*

date(*stringExp***)** The format of the result returned by *stringExp* should be in consistent with the date format in configuration information; if time is contained in the result, the time will not be converted

date(*year*,*month*,*day***)** Convert *year*,*month*,*day* of integer type to date type

Remark:

Convert the string *stringExp* or integer *year*,*month*,*day* to date

Parameters:

format	Format string
stringExp	String expression
year	Integer
month	Integer
day	Integer

Return value:

Date

Example:

- **date("1982-08-09")** **1982-08-09**
- **date("1982-08-09 10:20:30")** **1982-08-09**
- **date(1982,08,09)** **1982-08-09**
- **date(1982,-8,09)** **1981-04-09**
- **date(1982,18,09)** **6/9/1983**
- **date("12/28/1972","MM/dd/yyyy")** **1972-12-28**

Related concepts:

date(*datetimeExp*)
datetime(*datetimeExp*)
time(*datetimeExp*)
datetime()
time()

datetime()

datetime(datetimeExp)

Description:

Adjust the precision of datetime formula and then return

Syntax:

datetime(*datetimeExp***)**

Remarks:

Adjust the precision of *datetimeExp* and then return. By default, the default precision is the day

Parameters:

datetimeExp	datetime value

Options:

@m	Measure to minute

@s Measure to second

Return value:

Datetime value

Example:

- **datetime(now())** **2013-12-09 00: 00: 00**
- **datetime@m(now())** **2013-12-09 16:56: 00**
- **datetime@s(now())** **2013-12-09 16:56:45**

Related concepts:

date()

date(*datetimeExp*)

time(*datetimeExp*)

datetime()

time()

datetime()

Description:

Convert the string or long integer to date time

Syntax:

datetime(*string*{, *format* }**)** Convert the data type of *string* to date/time according to the format defined by *format*; if parameter *format* doesn't exist, the format of *string* of string type should be the same as the data type of date and time in configuration information

datetime(*long***)** Convert *long* of long integer type to date/time

datetime(*date*,*time***)** Concatenate date type *data* and *time* type data into data of date/time type

datetime(*y*,*m*,*d*,*h*,*m*,*s***)** Convert *y*,*m*,*d*,*h*,*m*,*s* of integer type to date/time data

Remark:

The format of *string* should match *format*

Parameters:

string	String
format	Format string
long	Long integer counted in microseconds
date	date type
time	time type
y	integer, year
m	integer, month
d	integer, day
h	integer, hour
m	integer, minute
s	integer, second

Return value:

Date time

Example:

- **datetime("2006-01-01 10:20:30")** **2006-01-01 10:20:30**
- **datetime("12/28/1972 10:23:43","MM/dd/yyyy hh:mm:ss")** **1972-12-28 10:23:43**
- **datetime("2006-01-01 10:20:30:111")** **2006-01-01 10:20:30**
- **datetime(12345)** **1970-01-01 08:00:12**
- **datetime(date("1982-08-09"),time("12:12:12"))** **1982-08-09 12:12:12**
- **datetime(2006,01,01,-10,-20,30)** **2005-12-31 13:40:30**

Related concepts:

date()
time()
date(*datetimeExp*)
datetime(*datetimeExp*)
time(*datetimeExp*)

day()

Description:

Get the day from a date

Syntax:

day(*dateExp***)**

Remark:

Get the day from the date *dateExp*

Parameters:

dateExp Date expression whose result is the date

Options:

@w Get the day of the week from the specified date. For Sunday, return 1; For Monday, return 2, and so on. By default, get the day of the month from the specified date.

Return value:

Integer

Example:

- **day(datetime("19800227","yyyyMMdd"))** **27**
- **day(datetime(12345))** **1**
- **day(datetime("2006-01-15 10:20:30"))** **15**
- **day@w(datetime("19800227","yyyyMMdd"))** **4**
- **day@w (datetime("2006-01-15 10:20:30"))** **1**

Related concepts:

year()
month()
hour()
minute()
second()
millisecond()

days()

Description:

Get the number of days of the year, quarter or month to which the specified date belongs

Syntax:

days(*dateExp***)**

Remark:

Get the number of days of the year, quarter or month to which the specified date *dateExp* belongs

Parameters:

dateExp	Expression whose result is a date or date time

Options:

@q	Get the number of days of the quarter to which the specified date belongs
@y	Get the number of days of the year to which the specified date belongs
	The default is the number of days of the month to which the specified date belongs

Return value:

Integer

Example:

- **days (datetime("19800227","yyyyMMdd"))** **29**
- **days (datetime("2006-01-15 10:20:30"))** **31**
- **days@y(datetime("19800227","yyyyMMdd"))** **366**
- **days@q(datetime("2006-01-15 10:20:30"))** **90**

decimal()

Description:

Convert a string or numeric value to a big decimal number.

Syntax:

decimal(*stringExp***)**

decimal(*numberExp***)**

Remark:

The calculation of *stringExp* must be a string composed of numbers and a decimal point at most.

The calculation of *numberExp* must be a numeric value which is less than or equal to 64 bit. For value more than 64 bits, you will have to use the *stringExp* to replace the *numberExp*.

Parameters:

stringExp	A string that consist of numbers, which may also contain a decimal point at most.
numberExp	The numeric value which is less than or equal to 64 bit.

Return value:

BigDecimal number

Example:

- **decimal("12345678901234567890123456789")** **12345678901234567890123456789**
- **decimal(1234567890123456)** **1234567890123456 (BigDecimal type)**

Related concepts:

float()
int()
long()
number()
string()

deq()

Description:

Judge if two dates are the same

Syntax:

deq(*datetimeExp1*,*datetimeExp2***)**

Remark:

Compare the two parameters of dateExp1 with dateExp2 to see if they are the same

Parameters:

datetimeExp1	Date or standard datetime format string such as yyyy-MM-dd HH:mm:ss, yyyy-MM-dd, or HH:mm:ss
datetimeExp2	Date or standard datetime format string such as yyyy-MM-dd HH:mm:ss, yyyy-MM-dd, or HH:mm:ss

Options:

@y	Precise to years
@q	Precise to quarters
@m	Precise to months
@t	Precise to ten-days
@w	Precise to weeks
	Precise to days by default

Return value:

Boolean

Example:

- **deq("1988-12-08","1988-12-07")** **false**
- **deq@y(date("1988-11-08"),date("1988-09-12"))** **true**
- **deq@m(date("1988-11-08"),date("1988-09-12"))** **false**
- **deq@q(date("1988-12-08"),date("1988-10-12"))** **true**
- **deq@t(date("1988-10-08"),date("1988-10-12"))** **false**
- **deq@w(date("1988-10-05"),date("1988-10-08"))** **true**

diff()

A.diff()

Description:

Execute difference operation between the first member and the other members of a sequence.

Syntax:

A.**diff()**

Remark:

The members in sequence *A* may also be sequence. Generate a new sequence by difference operation between the first member and the other members of *A*, so as to ensure the new sequence does not contain any member from the other member sequences.

The algorithm is to perform the difference operation between the first and the second member sequences, then, between the result of previous subtraction and the third member sequence, and so on.

Parameters:

A A sequence whose members are sequences

Return value:

The new sequence by difference operation between the first member and the other members of *A*

Example:

0	1		A	B
1-		1	Student	Math
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	PE
	1	9	Vincent	100
	1	10	Aaron	99
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==[{A2}(to(3)),{A9}(to(3))].diff()	

The student whose math score is among top 3 and PE score is not among the top 3

The value of **A15** is: **["David"]**

Related concepts:

A.union()

A.conj()

A.isect()

A.diff(x)

Description:

Perform a certain operation on a sequence so as to remove from the first sub-sequence of the sequence members of the other sub-sequences.

Syntax:

A.**diff(x)**

Remark:

Generally sequence *A* contains multiple sub-sequences. Compute x on each sub-sequence of *A* by loop to create a new sequence by getting the difference of the first sub-sequence and the other sub-sequences, to ensure that the new sequence doesn't include any member of the other sub-sequences.

The operation is to compute the difference of the first sub-sequence and the second one, then compute the difference of the result and the third sub-sequence, and so on and so forth.

Parameters:

A A sequence whose members are sequences

x an expression, "**~**" in which is used to reference the current member.

Return value:

A sequence

Example:

0	1		A	B
1-		1	Student	Math
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	PE
	1	9	Vincent	100
	1	10	Aaron	98
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==[{B2},{B9}].diff(~-10)	

A15 result: [85,77,73,65,55]

Related concepts:

A.diff()

Difference sequence

Description:

Generate a new sequence by subtracting members from a sequence.

Syntax:

A\B

A\x

Remark:

Generate a new sequence by subtracting the members (or single values) of *B* from sequence *A* in proper order

Parameters:

A an *n* sequence

B an *m* sequence or a single value; when it is a single value, it is regarded as **[***B***]**

Return value:

The new sequence by subtracting the members (or single values) of *B* from sequence *A* in proper order.

Example:

0	1		A	B
1-		1	Student	Math
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	PE
	1	9	Vincent	100
	1	10	Aaron	99
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==\{A2\}(to(3))\\{A9\}(to(3))	

The students whose math score is among the top 3 but the PE score is not among the top 3

The value of **A15** is: **["David"]**

Related concepts:

Concatenate sequence

Intersection sequence

Sequence Union

Alignment Arithmetic Operation

cmp()

dup()

A.dup()

Description:

Copy sequence

Syntax:

A.**dup()**

Remark:

Copy sequence *A*.

Parameter:

A sequence

Return value:

sequence

Example:

0	1		A
1-		1	[4,7,9]
	1	2	=A1.dup()
2		3	

A2 result: [4,7,9]

eq()

Description:

Judge if a sequence can be generated by swapping the positions of the members of another sequence.

Syntax:

A.eq(B)

Remark:

Judge if a sequence *A* can be generated by swapping the positions of the members of another sequence *B*.

Parameters:

A a sequence expression

B a sequence expression

Return value:

Boolean value.

Example:

0	1		A	B
1-		1	Student	PE
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83

	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	Math
	1	9	Lucy	100
	1	10	Vincent	99
	1	11	Mary	92
	1	12	David	88
	1	13	Charles	80
	1	14	Aaron	71
3		15	==[A2].eq([A9])	

Check if the student is weak in P.E. and good at other subjects. The return value is **true**

Empty sequence

Description:

A sequence having no member.

Remark:

A sequence having no member, for example: **[]**.

Escape character

Description:

A character that invokes an alternative interpretation on special characters in a string.

Syntax:

****s

Remark:

The special character such as **** (not an escape character, simply a backslash), **"**, **'**, newlines, etc, needs an alternative interpretation in a string, or it will be misunderstood.

The double quotation mark in **$[** *string***]** does not need to use the escape character; while the double quotation mark in **"***string***"** needs to use the escape character.

We support the following escape character:

\t

\r

\n

Parameters:

s The special character that needs an alternative interpretation such as **** (not an escape character, simply a backslash), **"**, **'**, newlines, etc.

Return value:

A string that contains the corresponding special character converted from the escape sequences.

Example:

- Common examples:

- **"\\"** backslash
- **"\n"** newlines

➢ Double quote in **$[]** does not need an escape character, while double quote in **""** does.

- **$[a"s]**
- **"a\"s"**

Related concepts:

String

eval()

Description:

Dynamically parse and compute the expression

Syntax:

eval(*StringExp* ,{*argExp*}**)**

Remark:

Use the result of *StringExp* as an expression string, parse it and compute it, then return its result. In the expression string, the value of the keyword **?** is the result of *argExp*. If there are more than one ?, then there may be more than one *argExp*, the *argExp* is corresponding to **?** one by one.

If the number of **?** is more than that of *argExp*, then the loop will start from the first *argExp*.

And also, you can use sequence number to specify the parameter for **?**, such as **eval("?2/?1", 3, 6)**, which means the first **?** correspond to the second parameter, the second **?** correspond to the first parameter, so the result is **2**.

Parameters:

StringExp	Expression string to be calculated
argExp	Parameter expression

Function keyword:

?	The key word in the expression string calculated from *StringExp*

Return value:

Result of expression. The data type will be determined by the expression

Example:

0	1		A	B	C	D	E	F
1-		1	1+3	4	3	6	1	
	1	2	==eval(A1)	==eval("?+5",B1)	==eval("(?+1)/?",C1,B1)	==eval("(?+?)*?",E1,C1)	==eval("?+?",C1)	==eval("?2/?1",C1,D1)

A2 result: 4

B2 result: The **?** is a key word, its value is **B1**, and the result is **9**

C2 result: The first **?** is **3**, and the second **?** is **4**, the result is **1.0**.

D2 result: The first **?** is **1**, the second **?** is **3**, and the third **?** is **1** again, the result is **4**.

E2 result: The result is **6** because the number of *argExp* is less than **?**, so it will use the *argExp* repeatedly.

F2 result: The first **?** correspond to the second parameter, the second **?** correspond to the first parameter, so the result is **2.0**.

exp()

Description:

Compute the powers of e

Syntax:

exp(*n*)

Remark:

Compute the *n* powers of e

Parameters:

n To specify the number of powers

Return value:

Numeric

Example:

- **exp(4.3)** **73.69979369959579**

Related concepts:

power(x, n)

f@o(...)

Description:

Introduce the common rules of functions.

Syntax:

***f*@*o*(...)** With various function options, a function can implement various functions. The basic format of function options is *f*@*o*(…), and "*o*" is the option of function *f*. For different function option.

Remark:

f Function name

@*o* Function option. Different options support different functions, and a same option has almost the same meaning in various functions.

(...) The input parameter of function; The separator of parameters can be colon **":"**, comma **","**, and semicolon **";"** and their priorities decrease from left to the right just as the introduction order above.

{...} The bracket for changing the level to which the parameters belong

Faccrint()

Description:

The function equals the Excel ACCRINT function

Syntax:

Faccrint(*first_interest*,*settlement*,*issue*;*rate*,*par***)**

Remark:

The function returns the accrued interest for a security that pays periodic interest. Specify the day count basis method **US (NASD) 30/360**.

Parameters:

first_interest	The security's first interest date
settlement	The security's settlement date, i.e. the date after the issue date by which a buyer must pay for the security
issue	The issue date of the security
rate	The security's annual coupon rate
par	The security's par value. If omitted, it takes the default value of zero for ￥100

Options:

@2 **Semi-annually**. It corresponds to the Excel *frequency* parameter.

@4 **Quaterly**. It corresponds to the Excel *frequency* parameter.

@1 Specify the day count basis method **Actual/Actual**. It corresponds to the Excel *basis* parameter.

@0 Specify the day count basis method **Actual/360**. It corresponds to the Excel *basis* parameter.

@5 Specify the day count basis method **Actual/365**. It corresponds to the Excel *basis* parameter.

@e Specify the day count basis method **European 30/360**. It corresponds to the Excel *basis* parameter.

Example:

Faccrint@2(date("2000-10-1"),date("2000-6-1"),date("2000-1-8"):0.1,2000)	**79.44444444444444**
Faccrint@1(date("2000-10-1"),date("2000-6-1"),date("2000-1-8"):0.1,2000)	**108.2089552238806**

Faccrintm()

Description:

The function equals the Excel ACCRINTM function

Syntax:

Faccrintm(*maturity*,*issue*;*rate*,*par***)**

Remark:

The function returns the accrued interest for a security that pays interest at maturity. **Annually**. Specify

the day count basis method **US (NASD) 30/360**.

Parameters:

maturity	The security's maturity date
issue	The security''s issue date
rate	The security's annual coupon rate
par	The security's par value. If omitted, it takes the default value of zero for 100 *yuan*

Options:

@1	Specify the day count basis method **Actual/Actual**. It corresponds to the Excel *basis* parameter.
@0	Specify the day count basis method **Actual/360**. It corresponds to the Excel *basis* parameter.
@5	Specify the day count basis method **Actual/365**. It corresponds to the Excel *basis* parameter.
@e	Specify the day count basis method **European 30/360**. It corresponds to the Excel *basis* parameter.

Example:

Faccrintm@1 (date("2001-6-15"),date("2001-4-1"):0.1,1000)	**20.54794520547945**

fact()

Description:

Compute the factorial of parameters

Syntax:

fact(*nExp***)**

Remark:

Compute the factorial of *nExp*

Parameters:

nExp	Data for which you want to compute the factorial

Return value:

Long integer (64 bit)

Example:

fact(10)	**3628800**

false

Description:

Logical constants. False value

Syntax:

false

Remark:

It can be used directly in the constant cell or expression.

Example:

0	1		A	B
1-		1	59	60
	1	2	==A1>B1	

The value of **A2** is **false**

Related concepts:

true

null

Fcoupcd()

Description:

The function equals the Excel COUPNCD or COUPNND function

Syntax:

Fcoupcd(*settlement*,*maturity*) Equivalent to Excel COUPNCD function. It returns the next coupon date after the settlement date for a security. **Annually**. Specify the day count basis method **US (NASD) 30/360**.

Fcoupcd@p(*settlement*,*maturity*) Equivalent to Excel COUPNND function. It returns the previous coupon date before the settlement date for a security. **Annually**. Specify the day count basis method **US (NASD) 30/360**.

Parameters:

settlement The security's settlement date

maturity The security's maturity date

Options:

@2 **Semi-annually**. It corresponds to the Excel *frequency* parameter.

@4 **Quaterly**. It corresponds to the Excel *frequency* parameter.

@1 Specify the day count basis method **Actual/Actual**. It corresponds to the Excel *basis* parameter.

@0 Specify the day count basis method **Actual/360**. It corresponds to the Excel *basis* parameter.

@5 Specify the day count basis method **Actual/365**. It corresponds to the Excel *basis* parameter.

@e Specify the day count basis method **European 30/360**. It corresponds to the Excel

basis parameter.

Example:

Fcoupcd@21(date("2007-1-25"),date("2008-11-15")) **2007-05-15**
Fcoupcd@p2(date("2008-3-15"),date("2008-11-3")) **2007-11-03**

Fcoups()

Description:

The function equals the Excel COUPNUN, COUPDAYS, COUPDAYBS or COUPDAYSNC function

Syntax:

Fcoups(*settlement*,*maturity*)	Equivalent to Excel COUPNUM function. It returns the number of coupons payable between a security's settlement date and maturity date, rounded up to the nearest whole coupon. **Annually**. Specify the day count basis method **US (NASD) 30/360**.
Fcoups@**d**(*settlement*,*maturity*)	Equivalent to Excel COUPDAYS function. It returns the number of days in a coupon period that contains the settlement date. **Annually**. Specify the day count basis method **US (NASD) 30/360**.
Fcoups@**b**(*settlement*,*maturity*)	Equivalent to Excel COUPDAYBS function. It returns the number of days from the beginning of a coupon's period to the settlement date. **Annually**. Specify the day count basis method **US (NASD) 30/360**.
Fcoups@**n**(*settlement*,*maturity*)	Equivalent to Excel COUPDAYSNC function. It returns the number of days from the settlement date to the next coupon date. **Annually**. Specify the day count basis method **US (NASD) 30/360**.

Parameters:

settlement	The security's settlement date
maturity	The security's maturity date

Options:

@2 **Semi-annually**. It corresponds to the Excel *frequency* parameter.

@4 **Quaterly**. It corresponds to the Excel *frequency* parameter.

@1 Specify the day count basis method **Actual/Actual**. It corresponds to the Excel *basis* parameter.

@0 Specify the day count basis method **Actual/360**. It corresponds to the Excel *basis*

parameter.

@5 Specify the day count basis method **Actual/365**. It corresponds to the Excel *basis* parameter.

@e Specify the day count basis method **European 30/360**. It corresponds to the Excel *basis* parameter.

Example:

```
Fcoups@2(date("2008-3-15"),date("2008-11-3"))    2
Fcoups@d2(date("2008-3-15"),date("2008-11-3"))   180.0
Fcoups@b1(date("2008-3-15"),date("2008-11-3"))   133.0
Fcoups@n(date("2008-3-15"),date("2008-11-3"))    228.0
```

Fdb()

Description:

The function equals the Excel DB function

Syntax:

Fdb(*cost,salvage,life,period,month***)**

Remark:

The function calculates the depreciation of an asset for a specified period, using the fixed-declining balance method

Parameters:

cost The initial cost of the asset

salvage The value of the asset at the end of the depreciation (asset residual value)

life The number of periods over which the asset is to be depreciated (expected useful life of the asset)

period The period number for which you want to calculate the depreciation (it must has the same unit with *life*)

month The number of the months used in the calculation of the first year of depreciation. If omitted, it takes the value of 12 by default.

Example:

```
Fdb(500000,100000,3,1,6)    103799.11308935669
```

Fddb()

Description:

The function equals the Excel DDB function

Syntax:

Fddb(*cost,salvage,life,period,factor***)**

Remark:

The function calculates the depreciation of an asset for a specified period, using the double-declining balance method, or some other user-defined method. Parameters should all be positive numbers.

Parameters:

cost	The initial cost of the asset
salvage	The value of the asset at the end of the depreciation (asset residual value)
life	The number of periods over which the asset is to be depreciated (expected useful life of the asset)
period	The period number for which you want to calculate the depreciation (It must has the same unit with *life*)
factor	The rate of balance declining. If omitted, it takes on the default value of 2 (specifying the double-declining balance method)

Example:

Fddb(100000,10000,3650,1) **54.794520547945204**

Fdisc()

Description:

The function equals the Excel DISC function

Syntax:

Fdisc (*settlement,maturity;pr,redemption***)**

Remark:

The function calculates the discount rate for a security. Specify the day count basis method **US (NASD) 30/360**.

Parameters:

settlement	The security's settlement date
maturity	The security's maturity date
pr	The security's price
redemption	The security's redemption value

Options：

@1 Specify the day count basis method **Actual/Actual**. It corresponds to the Excel *basis* parameter.

@0 Specify the day count basis method **Actual/360**. It corresponds to the Excel *basis* parameter.

@5 Specify the day count basis method **Actual/365**. It corresponds to the Excel *basis* parameter.

@e Specify the day count basis method **European 30/360**. It corresponds to the Excel *basis* parameter.

Example:

Fdisc@0(date("2001-2-15"),date("2001-6-10"),97.975,100) **0.06339130434782626**

Fduration()

Description:

The function equals the Excel DURATION function

Syntax:

Fduration(*settlement,maturity;coupon,yld***)**

Remark:

The function returns the modified duration of a security that pays periodic interest with an assumed par value of ￥100. The duration is the weighted average of the present value of the cash flows and is used to measure the sensitivity of the price of the security to the interest rates. **Annually**. Specify the day count basis method **US (NASD) 30/360**.

Parameters:

settlement	The security's settlement date
maturity	The security's maturity date
coupon	The security's annual coupon rate
yld	The security's annual yield

Options:

@2 **Semi-annually**. It corresponds to the Excel *frequency* parameter.

@4 **Quaterly**. It corresponds to the Excel *frequency* parameter.

@1 Specify the day count basis method **Actual/Actual**. It corresponds to the Excel *basis* parameter.

@0 Specify the day count basis method **Actual/360**. It corresponds to the Excel *basis* parameter.

@5 Specify the day count basis method **Actual/365**. It corresponds to the Excel *basis* parameter.

@e Specify the day count basis method **European 30/360**. It corresponds to the Excel *basis* parameter.

Example:

Fduration@21(date("2008-1-1"),date("2016-1-1"),0.08,0.09) **5.993774955545178**

fill()

Description:

To obtain a string by filling characters in it.

Syntax:

fill(*s,n***)**

Remark:

Get a string in which the number of *s* is *n*

Parameters:

s Source strings for making up a new string

n Number of source strings in a new string

Return value:

Character

Example:

- **fill("1 ",10)** **"1 1 1 1 1 1 1 1 1 1 "**
- **fill("a b",10)** **"a ba ba ba ba ba ba ba ba b"**

Fintrate()

Description:

The function equals the Excel INTRATE function

Syntax:

Fintrate(*settlement,maturity;investment,redemption***)**

Remark:

The function returns the interest rate for a security that pays interest at maturity. Specify the day count basis method **US (NASD) 30/360**.

Parameters:

settlement	The security's settlement date
maturity	The security's maturity date
investment	The initial amount invested into the security

redemption The security's redemption value

Options:

@1 Specify the day count basis method **Actual/Actual**. It corresponds to the Excel *basis* parameter.

@0 Specify the day count basis method **Actual/360**. It corresponds to the Excel *basis* parameter.

@5 Specify the day count basis method **Actual/365**. It corresponds to the Excel *basis* parameter.

@e Specify the day count basis method **European 30/360**. It corresponds to the Excel *basis* parameter.

Example:

Fintrate@0(date("2001-2-15"),date("2001-5-15");1000000,1014420) **0.058328089887640454**

Firr()

Description:

The function equals the Excel IRR function

Syntax:

Firr(*values,guess***)**

Remark:

The function calculates the internal rate of return for a series of cash flows represented by numeric values. The internal rate of return is the rate that makes the net present value from a particular investment equal to zero, that is, the current value of the returns received from the investment happens to be equal to the the investment cost.

Parameters:

values A sequence containing the values of a series of cash flows used to calculate the internal rate of return. It must contain at least one periodic payment (negative value) and at least one periodic income (positive value).

guess An estimated value for IRR function's calculation. If omitted, it will take on the default value of 0.1

Example:

Firr([-70000,12000,15000,18000,21000])	**-0.02124484827331908**
Firr([-70000,12000,15000,18000,21000,26000])	**0.08663094803666999**
Firr([-70000,12000,15000,18000,21000,26000],-0.1)	**0.08663094803655035**

float()

Description:

Convert a string or a number to a 64-bit double-precision floating-point number.

Syntax:

float(*stringExp***)**

float(*numberExp***)**

Remark:

The calculation of *stringExp* must be a string that consist of a number which is less than or equal to 64 bit. For value more than 64 bits, the result of **float(***stringExp***)** will be imprecise.

The calculation of *numberExp* must be a numeric value which is less than or equal to 64 bit. For value more than 64 bits, the result of **float(***numberExp***)** will be imprecise.

Parameters:

stringExp	The string expression you want to return as a double precision floating-point number.
numberExp	The number you want to return as a double precision floating-point number.

Return value:

64-bit double precision floating-point number

Example:

- **float("1234567")** **1234567.0**
- **float(1234567)** **1234567.0**

Related concepts:

decimal()

int()

long()

number()

string()

floor()

Description:

Truncate the data at the specified positions, and reject all the remaining part if any

Syntax:

floor(*numberExp*,{*nExp*}**)**

Remark:

Truncate the data *numberExp* at the specified position *nExp*, and reject all the remaining part if any

Parameters:

numberExp	Data to be intercepted
nExp	Integer; To specify the position from which to intercept, >0: Move the decimal point to the right for *nExp* places <0: Move the decimal point to the left for *nExp* places

=0: Indicate the current decimal places.

Return value:

Numeric

Example:

- **floor(3451231.234,0)** **3451231.0**
- **floor(3451231.234,-1)** **3451230.0**
- **floor(3451231.234,-2)** **3451200.0**
- **floor(3451231.234,1)** **3451231.2**
- **floor(3451231.234,2)** **3451231.23**

Related concepts:

ceil()

round()

Fmirr()

Description:

The function equals the Excel MIRR function

Syntax:

Fmirr(*values,finance_rate,reinvest_rate***)**

Remark:

The function returns the modified internal rate of return for a series of periodic cash flows, considering the cost of the investment and the interest on the reinvestment of cash

Parameters:

values	A sequence containing the values of payment (negative value) and income (positive value) at each of the regular periods . At least one negative value and at least one positive value must be contained.
finance_rate	The interest rate paid on the money invested
reinvest_rate	The interest rate paid on the reinvested cash which is the net income from the regular periods

Example:

Fmirr([-120000,39000,30000,21000,37000,46000],0.1,0.12) **0.1260941303659051**

Fnper()

Description:

The function equals the Excel NPER function

Syntax:

Fnper(*rate,pmt,pv,fv***)**

Remark:

The function returns the number of periods required to pay off a loan according to a specified periodic payment.

Parameters:

rate The interest rate per period; it is a fixed value

pmt The amount paid per period, which keeps unchanged during the whole period of paying off the loan. To omit it, *pv* must exist

pv The present value of the loan, a.k.a. the principal, that is the money that already exist when the payment for an investment (or a loan) begins, or the total amount of present values of a series of future payments

fv The future value of the loan, or the cash balance you hope to achieve after the final payment. If omitted, its value will be assumed as zero (for example the future value of a loan can be zero)

Options:

@t @t corresponds to Excel type parameter. If using the option, choose type 1; if not, choose type 0.

Example:

Fnper(0.06/12,-1200,150000) **196.65585756847307**

Fnpv()

Description:

The function equals the Excel NPV function

Syntax:

Fnpv(*rate,values***)** Equivalent to Excel NPVfunction

Fnpv(*rate,values,dates***)** Equivalent to Excel XNPVfunction

Remark:

The function returns the net present value of an investment, based on the discount rate and a series of future payments (negative value) and income (positive value). The net present value is the difference between the present value of future cash inflows and outflows from an investment and the amount of investment.

Parameters:

rate The discount rate over one period (that is equal to inflation rate and the rate of competitive investment); it is a fixed value

values A sequence of values representing payments and income that must occur at regular time intervals and at the end of each period

dates A sequence of dates corresponding to the array of cash payments. The first date of payment denotes the beginning of the payments for the investment

Example:

Fnpv(0.11,[-10000,3000,5000,6000])	**1034.201042097907 2**
Fnpv(0.11,[-10000,3000,5000,6000],[date("2008-01-01"),date("2008-05-23"),date("2009-03-10"),date("2009-5-15")])	**2497.061973416109**

Fpmt()

Description:

The function equals the Excel PMT function

Syntax:

Fpmt(*rate,nper,pv,fv***)**

Remark:

The function calculates each period's amount required to pay off an investment loan, based on a constant interest rate and the constant periodic payments.

Parameters:

rate The interest rate per period; it is a fixed value

nper The number of periods over which the investment (or loan) requires or is to be paid

pv The present value of the loan /investment, a.k.a. the principal, that is the money that already exist when the payment for an investment (or a loan) begins, or the total amount of present values of a series of future payments

fv The future value of the loan/investment, or the cash balance you hope to achieve after the final payment. If omitted, its value will be assumed as zero (for example the future value of a loan can be zero)

per The number of period in which the principal appears. Its value must between 1 and *nper*

Options:

@t @t corresponds to Excel type parameter. If using the option,

choose type 1; if not, choose type 0.

@i(*rate*,*nper*:*per*,*pv*,*fv*)	This option makes the function equivalent to Excel IPMT function. It enables to calculate the interest payment for a given period, with constant periodic payment and a constant interest rate
@p(*rate*,*nper*:*per*,*pv*,*fv*)	This option makes the function equivalent to Excel PPMT function. It enables to calculate the principal amount during a specific period of an investment or loan that is paid in constant periodic payments, with a constant interest rate

Example:

Fpmt@t(0.07/12, 10, 200000)	**-20647.264618755307**
Fpmt@i(0.1/12,36:1, 8000)	**-66.66666666666666**
Fpmt@p(0.07/12,12*10:1,120000)	**-693.3017506234848**

Fprice()

Description:

The function equals the Excel PRICE, PRICEDISC or PRICEMAT function

Syntax:

Fprice(*settlement*,*maturity*;*rate*,*yld*,*redemption*)	Equivalent to Excel PRICE function. It calculates the price per ¥100 par value of a security of that pays periodic interest. **Annually**. Specify the day count basis method **US (NASD) 30/360**.
Fprice@**d**(*settlement*,*maturity*;*discount*,0,*redemption*)	Equivalent to Excel PRICEDISC function. It calculates the price per ¥100 par value of a discounted security. Specify the day count basis method **US (NASD) 30/360**.
Fprice@**m**(*settlement*,*maturity*,*issue*;*rate*,*yld*)	Equivalent to Excel PRICEMAT function. It calculates the price per ¥100 par value of a security that pays interest at maturity. Specify the day count basis method **US (NASD) 30/360**.

Remark:

The function returns the price of a security that pays periodic interest, or of a discounted security, or of a security that pays periodic interest

Parameters:

settlement	The security's settlement date

maturity	The security's maturity date
rate	The security's annual coupon rate
yld	The security's annual yield
redemption	The security's redemption value
discount	The security's discount rate
issue	The security's issue date

Options:

@2	**Semi-annually**. It corresponds to the Excel *frequency* parameter.
@4	**Quaterly**. It corresponds to the Excel *frequency* parameter.
@1	Specify the day count basis method **Actual/Actual**. It corresponds to the Excel *basis* parameter.
@0	Specify the day count basis method **Actual/360**. It corresponds to the Excel *basis* parameter.
@5	Specify the day count basis method **Actual/365**. It corresponds to the Excel *basis* parameter.
@e	Specify the day count basis method **European 30/360**. It corresponds to the Excel *basis* parameter.

Example:

Fprice@m(date("2001-2-15"),date("2002-4-13"),date("2000-11-11");0.061,0.061)	**99.89464887553818**
Fprice@d1(date("2001-2-15"),date("2002-4-13"),0.4,0,0.061)	**0.03278958904109588**

Frate()

Description:

The function equals the Excel RATE function

Syntax:

Frate(*nper,pmt,pv,fv,guess***)**

Remark:

The function calculates the interest rate required to pay an investment. Both the interest rate of the constant periodic payment and that of the one-off payment can be calculated.

Parameters:

nper	The number of periods over which the investment or loan is to be paid
pmt	The payment amount per period, including the principal and the interest
pv	The present value of the loan /investment, a.k.a. the principal, that is the money that already exist when the payment for an investment (or a loan) begins, or the

total amount of present values of a series of future payments

fv The future value of the loan/investment, or the cash balance you hope to achieve after the final payment. If omitted, it will take on the default value of zero

guess An estimated interest rate. If omitted, it will take on the assumed value of 10%. Both *guess* and *nper* must use the same unit.

Options:

@t @t corresponds to Excel type parameter. If using the option, choose type 1; if not, choose type 0.

Example:

Frate(12*6, -2200, 100000) **0.013798002390137705**

Freceived()

Description:

The function equals the Excel RECEIVED function

Syntax:

Freceived(*settlement,maturity;inverstment,discount***)**

Remark:

The function returns the amount received at maturity for a security

Parameters:

settlement The security's settlement date

maturity The security's maturity date

investment The initial amount invested into the security

discount The security's discount rate

Options:

@1 Specify the day count basis method **Actual/Actual**. It corresponds to the Excel *basis* parameter.

@0 Specify the day count basis method **Actual/360**. It corresponds to the Excel *basis* parameter.

@5 Specify the day count basis method **Actual/365**. It corresponds to the Excel *basis* parameter.

@e Specify the day count basis method **European 30/360**. It corresponds to the Excel *basis* parameter.

Example:

Freceived@0(date("2001-2-15"),date("2001-5-15"),1000000,0.0575) **1014420.2658626447**
Freceived@5(date("2001-2-15"),date("2001-5-15"),1000000,0.0575) **1014219.9190013407**

Fsln()

Description:

The function equals the Excel SLN function

Syntax:

Fsln(*cost,salvage,life***)**

Remark:

The function returns the straight-line depreciation of an asset for each period. The depreciation for each period is the same.

Parameters:

cost The initial cost of the asset
salvage The value of the asset at the end of the depreciation (also known as asset residual value)
life The number of periods over which the asset is to be depreciated (sometimes called expected useful life of the asset)

Example:

Fsln(300000, 20000,8) **35000.0**

Fsyd()

Description:

The function equals the Excel SYD function

Syntax:

Fsyd(*cost,salvage,life,period***)**

Remark:

The function returns depreciation of an asset for a specified period, using the sum-of-years' digits method.

Parameters:

cost The initial cost of the asset

salvage	The value of the asset at the end of the depreciation (also known as asset residual value)
life	The number of periods over which the asset is to be depreciated (sometimes called expected useful life of the asset)
period	The period for which the asset's depreciation is calculated (it must use the same unit as *life*)

Example:

Fsyd(500000, 5000,10, 1) **90000.0**

Fv()

Description:

The function equals the Excel FV function

Syntax:

Fv(*rate,nper,pmt,pv***)**

Remark:

The function calculates the future value of an investment (i.e. the sum of principal amount and interest obtained after the investment ends) with periodic constant payments and a constant interest rate.

Parameters:

rate	The interest rate per period; it is a fixed value
nper	The number of periods over which the investment (or loan) requires or is to be paid
pmt	The amount paid per period, which keeps unchanged during the whole period of paying off the loan. To omit it, *pv* must exist
pv	The present value of the loan /investment, a.k.a. the principal, that is the money that already exist when the payment for an investment (or a loan) begins, or an accumulated sum of present values of a series of future payments

Option:

@t	**@t** corresponds to the Excel *type* parameter. If using the option, choose type 1; if not, choose type 0.
@p(*rate,nper,pmt,fv*)	This option makes the function equivalent to Excel PV function. It enables to calculate the present value of an investment that is the total amount of a series of future payments. For example the amount of borrower's borrowed money is the present value of the loan delivered by the lender.

Example:

Fv@t(0.07, 30,-2000,0,1)	**202146.08273281078**
Fv@p(0.067/12,12*25,500,0)	**-72700.0451136414**

Fvdb()

Description:

The function equals the Excel VDB function

Syntax:

Fvdb(*cost,salvage,life,start_period,end_period,factor***)**

Remark:

The function calculates the depreciation of an asset for a specfied period, using variable declining balance method, or double-declining balance method or another user-defined method

Parameters:

cost	The initial cost of the asset
salvage	The value of the asset at the end of the depreciation (also known as asset residual value)
life	The number of periods over which the asset is to be depreciated (sometimes called expected useful life of the asset)
start_period	The starting period for which you want to calculate the depreciation. It must use the same unit as *life.*
end_period	The ending period for which you want to calculate the depreciation. It must use the same unit as *life.*
factor	The rate of depreciation. If omitted, it takes on the default value of 2, specifying the double-declining balance method

Option:

@s	Do not switch to the straight-line depreciation method when depreciation is greater than the declining balance calculation

Example:

Fvdb(100000, 10000,10, 0,1)	**20000.0**
Fvdb@s(100000, 10000,120, 6,18,1.5)	**12992.03902742642**

Fyield()

Description:

The function equals the Excel YIELD, YIELDDISC or YIELDMAT function

Syntax:

Fyield(*settlement*,*maturity*;*rate*,*pr*,*redemption*)	Equivalent to Excel YIELD function. It calculates the yield rate of a security. **Annually**. Specify the day count basis method **US (NASD) 30/360**.
Fyield@d(*settlement*,*maturity*;0,*pr*,*redemption*)	Equivalent to Excel YIELDDISC function. It calculates the annual yield rate of a discounted security. Specify the day count basis method **US (NASD) 30/360**.
Fyield@m(*settlement*,*maturity*,*issue*;*rate*,*pr*)	Equivalent to Excel YIELDMAT function. It calculates the yield rate of a security that pays interest at maturity. Specify the day count basis method **US (NASD) 30/360**.

Remark:

The function returns the yield rate of a security that pays periodic interest

Parameters:

settlement	The security's settlement date
maturity	The security's maturity date
rate	The security's annual coupon rate
pr	The security's price
redemption	The security's redemption value
issue	The security's issue date

Options:

@2	**Semi-annually**. It corresponds to the Excel *frequency* parameter.
@4	**Quaterly**. It corresponds to the Excel *frequency* parameter.
@1	Specify the day count basis method **Actual/Actual**. It corresponds to the Excel *basis* parameter.
@0	Specify the day count basis method **Actual/360**. It corresponds to the Excel *basis* parameter.
@5	Specify the day count basis method **Actual/365**. It corresponds to the Excel *basis* parameter.
@e	Specify the day count basis method **European 30/360**. It corresponds to the Excel *basis* parameter.

Example:

Fyield@2(date("2008-2-15"),date("2016-11-15");0.0575,95.04287,100)	**0.06500000688109805**
Fyield@d0(date("2008-2-16"),date("2008-3-1");0,99.795,100)	**0.05282257198685834**
Fyield@m(date("2008-3-15"),date("2008-11-3"),date("2007-11-8");0.0625,100.0123)	**0.060954333691538576**

gcd()

Description:

Return the greatest common divisor

Syntax:

gcd(x_i,…)
gcd(A)

Remark:

Calculate the greatest common divisor of members of [x_1, x_2,…]

Parameter:

A	Sequence
x_i	A numeric expression that will be ignored if its value is not a number

Return value:

The greatest common divisor

Example:

gcd(7,1)	**1**

Hexadecimal long integer

Description:

Hexadecimal long integer

Syntax:

0x12

Remark:

Those starting with "0x" are the hexadecimal long integers

Example:

- **0x2345** **9029**

Related concepts:

Long integer

hour()

Description:

Get the hour from a specified time

Syntax:

hour(*datetimeExp***)**

Remark:

Get the hour from the specified time *datetimeExp*

Parameters:

datetimeExp Expression whose result is a time or date time,

Return value:

Integer

Example:

- **hour("1983-12-15")** **0**
- **hour("1983-12-15 10:30:25")** **10**
- **hour(datetime("2006-01-15 13:20:30"))** **13**

Related concepts:

year()
month()
day()
minute()
second()
millisecond()

Identifier

Description:

Identifier (commonly referred to as variable name) is a defined identifier or defined variable name.

Syntax:

string1

'*string2*'

Remark:

Normally the variable name can be used directly without having to be defined specially. But if the variable name contains the space, equal sign, and other misleading characters, then the variable name will need to be defined specially. This is same to the principle of handling space in DOS command.

Parameters:

string1 Ordinary identifier

'string2' If the string contains any misleading characters such as **=**, space, **+**, and **-** , the string must be enclosed in single quotation marks.

Return value:

Variable using *string1* or '*string2*' as variable name.

Example:

0	1		A	
1-		**1**	**=arg1=5**	Common identifier
	1	**2**	**='a b'=4**	Identifier that contains a space

2		3	='a=b'=3	Identifier that contains equal signs
3		4	=arg1	5
4		5	='a b'	4
5		6	='a=b'	3

if()

Description:

Calculate the boolean expression from left to right. if anyone is true, return the true as ultimate return; otherwise return default value or false value.

Syntax:

if(*a***)**	If *a* is *true*, then return *true*. Otherwise, return *false*
if(*a*,*b*,*c***)**	If *a* is **true**, then return *b*, otherwise, return *c*, *c* is *null* by default.
if(*x*1:*y*1,…,*xk*:*yk*;*y***)**	**if(***x*1,*y*1,**if(***x*2,*y*2,…,**if(***xk*,*yk*,*y***)))**

Remark:

This function starts calculating from the left to the right. x_k is a boolean expression, if x_k is true, then return the result of y_k, and the calculation will be terminated; If x_k is false, then calculate the next x_k. If none of the boolean expression x_k is true, and there is a default expression y, then return the result of y, otherwise return **null**.

Parameters:

a	Boolean expression
b	True value. If the result of *a* is true, then return the calculation of *b*
c	False value. If the result of *a* is false, then return the calculation of *c*
x_k	Boolean expression
y_k	True value.
y	Default expression.

Return value:

The data type is uncertain, which is determined by the expression. If the corresponding expression is absent, then return null.

Example:

0	1		A	B	C	D
1-		1	85	300		
	1	2	==if(B1>A1,"Truth","Fallacy")	==if(A1>90:"Excellent",A1>80:"Good",A1>60:"Passed","Failed")	==if(B1>100:,B1>90:"Excellent",B1>80:"Good",B1>60:"Passed","Failed")	==if(B1>100)

The result of **A2** is: **Truth**

The result of **B2** is: **Good**

The result of **C2** is: **null**

The result of **D2** is: **true**

Related concepts:

case()

in()

ifa()

Description:

To judge if an object is a sequence.

Syntax:

ifa(*x***)**

Remark:

Judge if *x* is a sequence

Parameters:

x the object to be judged

Return value:

A boolean value

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
2		3	==ifa([B2:F2])	==ifa(98)				

The value of **A3** is: **true**

The value of **B3** is: **false**

ifdate()

Description:

Judge if the parameter is a date or date time.

Syntax:

ifdate(*exp***)**

Remark:

Judge if the parameter *exp* is a date or date time.

Parameters:

exp Data expression of any type

Return value:

Boolean

Example:

- **ifdate("2006-10-10")** **false**
- **ifdate(date("2006-10-10"))** **true**
- **ifdate(date("2006-10-10 10:20:30"))** **true**
- **ifdate("20061010")** **false**
- **ifdate("10:20:30")** **false**

Related concepts:

iftime()

ifn()

A.ifn()

Description:

Get the first non-null member in a sequence.

Syntax:

A.**ifn()** Equivalent to **ifn**(x_1,…,x_n)

Remark:

Get the first non-null member in the sequence *A*.

Parameters:

A an *n* sequence

Return value:

The first non-null member in the sequence

Example:

0	1		A	B	C	D	
1-		1	Student	PE	Math	English	
	1	2	Aaron	87		98	
	1	3	Charles		99	80	
	1	4	David	75	92		
	1	5	Mary	93		81	
	1	6	Vincent	75	90		
	1	7	Lucy	65	71	89	
	1	8	Petter	50	89		
2		9		==\{B2\}.sort().ifn()	==\{C2\}.sort().ifn()	==\{D2\}.sort().ifn()	Get the lowest score of each subject
3		10		==ifn(87,null,75,93,75,65,50)			

B9 result: 50

C9 result: 71

D9 result: 80

B10 result: 87

A.ifn(x)

Description:

Compute x with each member of the sequence and return the first non-null member of the new sequence

Syntax:

A.**ifn**(*x*) Equivalent to *A*.(*x*).**ifn**()

Remark:

Compute x on sequence A by loop and return the first non-null member of the new sequence

Parameters:

A a sequence

x an expression, "**~**" in which is used to reference the current member.

Return value:

The first non-null member of the new sequence

Example:

0	1		A	B
1-		1	Student	Math
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	PE
	1	9	Vincent	100
	1	10	Aaron	98
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==\{B2\}.ifn(~-10)	==\{B2\}.(~-10).ifn()

A15，B15 results: 88

Related concepts:

A.ifn()

ifnumber()

Description:

Judge if the parameter is a number.

Syntax:

ifnumber(*Exp*)

Remark:

Judge if the parameter *Exp* is a number

Parameters:

Exp Data expression of any type

Return value:

Boolean

Example:

- **ifnumber("abc")** **false**
- **ifnumber("1234")** **false**
- **ifnumber(1234)** **true**
- **ifnumber("1234sss")** **false**

Related concepts:

ifstring()

ifstring()

Description:

Judge if the parameter is a string

Syntax:

ifstring(*Exp* **)**

Remark:

Judge if the parameter *Exp* is a string

Parameters:

Exp Data expression of any type

Return value:

Boolean

Example:

- **ifstring("abc")** **true**
- **ifstring(1234)** **false**
- **ifstring("1980-01-01")** **true**
- **ifstring(date("1980-01-01"))** **false**

Related concepts:

ifnumber()

iftime()

Description:

Judge if the parameter is a time.

Syntax:

iftime(*exp***)**

Remark:

Judge if the parameter *exp* is a time.

Parameters:

exp Data expression of any type.

Return value:

Boolean

Example:

- **iftime("10:20:30")** **false**

- **iftime(time("10:20:30"))** **true**
- **iftime("2006-10-10")** **false**
- **iftime("2006-10-10 10:20:30")** **false**
- **iftime("20061010")** **false**

Related concepts:

ifdate()

in()

in()

Description:

Based on the passed-in parameter, judge if the Parameter 1 is between the Parameter 2 and Parameter 3.

Syntax:

in(*x*,*a*:*b*) To judge if *x* is between *a* and *b*. The default include *a*, *b*.

in(*x*,*a*) equal to in(x a:a)

in(*x*,*a*:) equal to in(x, a:infinity)

in(*x*,:*b*) equal to in(x, infinitely small: b)

Remark:

To judge if *x* is between *a* and *b*.

Parameters:

x an expression, the result of which must be a numeric string or a number.

a an expression, the result of which must be a numeric string or a number.

b an expression, the result of which must be a numeric string or a number.

Options:

@l Do not include *a*

@r Do not include *b*

@b if *x*<*a* return -1
if *b*<*x* return 1
Otherwise return 0

Return value:

Boolean

Example:

- **in(4,5:6)** **false**
- **in(4,3:6)** **true**
- **in(4,4:6)** **true**
- **in@l(5,5:6)** **false**
- **in@r(6,5:6)** **false**
- **in@b(5,6:9)** **-1**

Related concepts:

if()

case()

A.in()

Description:

Judge if a sequence contains another sequence

Syntax:

A.**in**(*B*)

Remark:

Judge if the sequence B contains the sequence A. Return true if the sequence B contains the sequence A, otherwise, return false.

Parameters:

A Sequence object

B Sequence object

Return value:

true/false

Example:

0	1		A
1-		1	=[2,3,4,5].in([1,2,3,4,5,6]) true

Related concepts:

Instant calculation cell

Description:

Define an instant calculation expression. This cell is the instant calculation cell.

Syntax:

=*x*

Remark:

Define a calculation expression *x*. If started with =, then perform the instant calculation without auto recalculation. That is, *x* will not be recalculated automatically even if other cells are referenced in the *x*, and the values of referenced cells have changed.

Parameters:

x Calculation expression

Return value:

Calculation of expression *x*

Example:

- **=1+1** Cell value is 2
- **=A1+1** Suppose the cell value of A1 is 2, then this cell returns the value of 3. If the cell value of A1 becomes 3, then this cell value will not be recalculated, and the cell

value is still the 3.

int()

Description:

This function is used to obtain the integer part of a numeric value from a numeric string or a number, and convert its data type to 32-bit integer.

Syntax:

int(*valueExp***)**

Remark:

The result of *valueExp* must be a numeric string or a number.

Parameters:

valueExp an expression, the result of which must be a numeric string or a number.

Return value:

32-bit integer

Example:

- **int("33")** **33**
- **int("33.999d")** **33**
- **int(1.5*1.5)** **2**
- **int(25.67)** **25**

Related concepts:

float()
decimal()
long()
number()
string()

Intersection sequence

Description:

Generate a new sequence which is composed of common members from two sequences.

Syntax:

A^*B*

Remark:

Generate a new sequence which is composed of members both in *A* and *B*

Parameters:

A an *n* sequence

B an *m* sequence

Return value:

The new sequence which is composed of members both in *A* and *B*

Example:

0	1		A	B
1-		1	Student	English
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	Math
	1	9	Vincent	100
	1	10	Aaron	99
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==\{A2\}(to(3))^\{A9\}(to(3))	

The student whose math and English scores are all among the top 3

The value of **A15** is: **["Aaron","Charles"]**

Related concepts:

Concatenate sequence

Difference sequence

Sequence Union

Alignment Arithmetic Operation

cmp()

interval()

Description:

Compute the interval between two date time data

Syntax:

interval (*datetimeExp1*,*datetimeExp2***)**

datetimeExp1- datetimeExp2 **interval (***datetimeExp1*,*datetimeExp2***)**

Remark:

Compute the days between two date time data *datetimeExp1* and *datetimeExp2*

Parameters:

datetimeExp1 The date expression whose result is a date, time or date time

datetimeExp2 The date expression whose result is a date, time or date time

Options:

@y Compute the years between two date time data

@q	Compute the quarters between two date time data
@m	Compute the months between two date time data
@s	Compute the seconds between two date time data
@ms	Compute the milliseconds between two date time data
@r	Compute the timespan between two date time data and return Real value
	The default is to compute the days between two date time data

Return value:

Integer

Example:

- **interval(datetime("19800227","yyyyMMdd"),datetime("1983-02-27 00:00:45"))** **1096**
- **interval@y(datetime("19800227","yyyyMMdd"),datetime("1983-02-27 00:00:45"))** **3**
- **interval@q(datetime("19800227","yyyyMMdd"),datetime("1983-02-27 00:00:45"))** **12**
- **interval@m(datetime("19800227","yyyyMMdd"),datetime("1983-02-27 00:00:45"))** **36**
- **interval@s(datetime("19800227","yyyyMMdd"),datetime("1980-02-27 00:00:45"))** **45**
- **interval@s ("1972-11-08 10:20:30","1972-11-08 10:30:50")** **620**
- **interval@ms(datetime("19800227","yyyyMMdd"),datetime("1980-02-27 00:00:45"))** **45000**
- **interval@ms("1972-11-08 10:20:30","1972-11-08 10:30:50")** **620000**
- **interval@r(datetime("19800227","yyyyMMdd"),datetime("1980-02-27 00:00:45"))**
  5.208333333333333E-4
- **interval@r("1972-11-08 10:20:30","1973-11-08 10:30:50")**
  365.00717592592594
- **datetime("19850227","yyyyMMdd")-datetime("1983-02-27 00:00:45")** **731**

inv()

A.inv()

Description:

Using the members of p as the rankings of A, adjust the order of A according to p, and return A after the adjustment.

Syntax:

A.**inv(***p***)**

Remark:

Using the members of *p* as the rankings of *A*, adjust the order of *A* according to *p*, and return *A* after the adjustment

Parameters:

p	an integer sequence, members of which is the rankings of *A*, so the number of its members is the same as that of *A* , and it is a unique *n* sequence (*n*=*A*.**len()**)
A	a sequence or a record sequence

Return value:

Sequence *A* after the adjustment

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	90	98	80	98
2		3	==[B2:F2].inv([2,1,4])					

A3 result: [90,87,null,98,null]

Note:

p must be a unique *n* integer sequence and *n* must be equal to *A*.**len()**

If *p* has duplicate members or its number of the members is not equal to that of *A*, return null.

If the member value of *p* exceeds the sequence number of *A*, return null

The case of tied rankings is not processed

Related concepts:

p.inv(k)

p.inv()

Description:

To compute the inverse ISeq of an ISeq.

Syntax:

p.**inv(***k***)**

Remark:

Return the sequence numbers of the numbers from **1** to *k* in integer sequence *p*, and return **0** for the numbers do not exist in *p*.

Parameters:

p an integer sequence

k an integer, *k* is *p*.**len()** by default

Return value:

The integer sequence composed of the sequence numbers of the numbers from **1** to *k* in integer sequence *p*

Example:

0	1		A	B
1-		1	Student	Age
	1	2	Aaron	3
	1	3	Petter	1
	1	4	Linda	6
	1	5	Lusa	4
2		6	==[B2].inv(4)	

A3 result: [2,0,1,4]

Of the four numbers of 1, 2, 3, and 4, the respective sequence numbers of 1, 3, and 4 in the sequence **[B2:B5]** are 2,1,and 4. Because the number of 6 does not exist in the sequence **[B2:B5]**, the sequence number of 6 is

0. The **[2,0,1,4]** will be returned.

Related concepts:

A.inv(p)

isalpha()

Description:

Judge if a string is composed of letters

Syntax:

isalpha(*s* **)**

Remark:

Judge if the string *s* is composed of letters. Or if *s* is an integer, look it as an ascii code, judge if its corresponding character is a letter.

Parameters:

s String/ Integer expression

Return value:

Boolean

Example:

- **isalpha ("abc")** **true**
- **isalpha (97)** **true**
- **isalpha("@#$")** **false**
- **isalpha("1@23")** **false**
- **isalpha("a@23")** **false**

Related concepts:

isdigit()

isdigit()

Description:

Judge if a string is composed of numbers.

Syntax:

isdigit (*string***)**

Remark:

Judge if the string *string* is composed of numbers. Or if *string* is an integer, look it as an ascii code, judge if its corresponding character is a number.

Parameters:

string String/Integer expression

Return value:

Boolean

Example:

- **isdigit ("123")** **true**

- isdigit (123) false
- isdigit ("abc") false
- isdigit ("123ss") false

Related concepts:

isalpha()

isect()

A.isect()

Description:

Compute the intersection of all the member sequences of a sequence.

Syntax:

A.**isect()**

Remark:

The members in sequence *A* may also be sequence. Generate a new sequence composed of members exist in all the member sequences.

Parameters:

A A sequence whose members are also sequences

Return value:

A sequence

Example:

0	1		A	B
1-		1	Student	English
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	80
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	Math
	1	9	Vincent	100
	1	10	Aaron	99
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==[{A2}(to(3)),{A9}(to(3))].isect()	

The student whose math and English scores are all among the top3

The value of **A15** is: **["Aaron","Charles"]**

Related concepts:

A.union()

A.diff()

A.conj()

A.isect(x)

Description:

Compute x with each member of the sequence whose members are sequences, and then perform intersection operation between members of the new sequence

Syntax:

A.**isect(x)**

Remark:

Members of sequence *A* are also sequences. Compute x on sequence *A* and create a new sequence composed of all common members of the sub-sequences

Parameters:

A A sequence whose members are also sequences

x an expression, "**~**" in which is used to reference the current member.

Return value:

A sequence

Example:

0	1		A	B
1-		1	Student	Math
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	PE
	1	9	Vincent	100
	1	10	Aaron	98
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==[{B2},{B9}].isect(~-10)	

A15 result: [88]

Related concepts:

A.isect()

islower()

Description:

Judge if a string is composed of letters in lower case.

Syntax:

islower (*string***)**

Remark:

Judge if the string *string* is composed of letters in lower case. Or if *string* is an integer, look it as an ascii code, judge if its corresponding character is a letter in lower case.

Parameters:

string String expression/ Integer expression

Return value:

Boolean

Example:

- **islower ("dgfdsgf")** **true**
- **islower (97)** **true**
- **islower ("dsfaAFD")** **false**
- **islower ("97ffdsf")** **false**

Related concepts:

isupper()

isupper()

Description:

Judge if a string is composed of letters in upper case.

Syntax:

isupper (*string* **)**

Remark:

Judge if the string *string* is composed of letters in upper case. Or if *string* is an integer, look it as an ascii code, judge if its corresponding character is a letter in upper case.

Parameters:

string String expression/ Integer expression

Return value:

Boolean

Example:

- **isupper ("ADSFDGKJ")** **true**
- **isupper (85)** **true**
- **isupper ("SDsdsSDAS")** **false**
- **isupper ("8ASDS7")** **false**

Related concepts:

islower()

L#

Description:

At the level L, get the sequence number of the current cell among its peer cells.

Syntax:

#

Remark:

Equivalent to **ord**(*current cell*,*L*). In the range of level L, get the sequence number of current cell among its peer cells

Parameters:

None

Return value:

At the level L, the sequence number of the current cell among its peer cells

Example:

0	1	2		A	B	C	D	E	F
1-			1	Dept	ID	Name	Birthday	Salary	
	1-		2	Admin					=F1#
		1	3	Admin	1	Mike	1968-12-08	8000	=F1#
		1	4	Admin	4	Andy	1968-09-19	6000	=F1#
	2		5						
	1-		6	R&D					=F1#
		1	7	R&D	2	Jake	1962-02-19	9000	=F1#
		1	8	R&D	3	Lucy	1973-08-30	10000	=F1#
		1	9	R&D	5	Jim	1965-03-04	4000	=F1#
	2		10						

F2,F6 result: 1,2, equivalent to **ord**(F2,F1), and **ord**(F6,F1) respectively

F3-F4,F7-F9 result: 1,2,3,4,5. Take F3 for example, F1# is equivalent to **ord**(F3,F1)

L##

Description:

In the range of the level L, get the number of peer cells of the current cell

Syntax:

##

Remark:

In the range of the level L, get the number of peer cells of the current cells, which is equivalent to **num** (current cell, *L*)

Parameters:

None

Return value:

In the range of the level L, get the number of peer cells of the current cells

Example:

0	1	2		A	B	C	D	E	H
1-			1	Dept	ID	Name	Birthday	Salary	
	1-		2	Admin					=H1##
		1	3	Admin	1	Mike	1968-12-08	8000	= H1##
		1	4	Admin	4	Andy	1968-09-19	6000	= H1##
	2		5						
	1-		6	R&D					= H1##
		1	7	R&D	2	Jake	1962-02-19	9000	= H1##
		1	8	R&D	3	Lucy	1973-08-30	10000	= H1##
		1	9	R&D	5	Jim	1965-03-04	4000	= H1##
	2		10						

H2,H6 result: 2, 2; equivalent to num(H2, H1), and num(H6, H1) respectively

H3,H7 result: 5, 5; take H3 for an example, H1## is equivalent to num(H3, H1)

L[A;x]

Description:

In the range of cell L, the peer cells of A are clustered. Locate the peer cells that are relatively x away from the peer cells of the current A.

Syntax:

L[A;x]

Remark:

In the range of cell L and the cell set composed of the peer cells of A, locate the peer cells x away from the peer cells of A and return null if out of range. In case there is no peer cells of A in the current records, then shift x away from the first member of the set of peer cells. The A is only used to indicate the peer cells. Therefore, the A is not necessarily required in the range of cell L.

Parameters:

L Cell

A Cell. only representing the peer cells

x Integer to represent the offset of peer cells relative to the current *A*

Return value

Cell value

Example:

0	1	2		A	B	C	D	E	F
1-			1	Dept	ID	Name	Birthday	Salary	=A1[B7;1]
	1-		2	Admin					=A2[B7;1]
		1	3	Admin	1	Mike	1968-12-08	8000	=A2[B3;-1]

		1	4	Admin	4	Andy	1968-09-19	6000	=A2[B4;-1]
	2		5						
	1-		6	R&D					=A6[B9;1]
		1	7	R&D	2	Jake	1962-02-19	9000	=A6[B7;-1]
		1	8	R&D	3	Lucy	1973-08-30	10000	=A6[B8;-1]
		1	9	R&D	5	Jim	1965-03-04	4000	=A6[B9;-1]
	2		10						

F1 result:4. In the range of A1, the peer cells of B7 are B3, B4, B7, B8, and B9. It is to move downward a cell relative to the B3, and the result returned is the cell value of B4.

F2 result:4. In the range of A2, the peer cells of B7 are B3 and B4. It is to move a cell downward relative to B3, and then the result returned is the cell value of B4.

Similarly, the result of F6 is 3.

F4 result: 1. In the range of A2 cell, the peer cell of B4 is B3 and B4. It is to move a cell upward relative to the current cell of B4, and then the cell value of B3 will be returned.

Similarly, F3, F7-F9 results are null, null, 2, and 3 respectively. Of which, F3 and F7 exceeds the boundary and thus return null.

L{A;a:b}

Description:

In the range of cell L, the peer cells of A are relatively clustered. The starting point and the ending point are the cells a and b away from the peer cells of the current A, respectively. The return value is a cell set composed of peer cells in this section.

Syntax:

L{A;a:b}

Remark:

In the range of cell L, in the cell set of peer cells of A, the starting point and the end point are respectively a and b away from the peer cells of the current A. Then, a cell set composed of peer cells in this section will be returned. If there is no peer cells of A among the current records, then shift to the cells relative to the first member of the set of peer cells. The A can only be used to represent the peer cells. Therefore, the A is not necessarily in the range of cell L.

Parameters:

L Cell . If omitting L, then it is interpreted as the current cell and the higher level of the parent cells of *A*

A Cell . Only represent the cell at the same level and in the same column

a Integer. Only represent the offset relative to the peer cells of the current *A*. In case of omitting or out-of-range, then the first member of *L{A}* will be regarded as the starting point.

b Integer. Represent the offset relative to the peer cells of the current *A*. In case of omitting or out-of-range, then the last member of *L{A}* will be regarded as the end point.

Return value

A sequence composed of cell value

Example:

0	1	2		A	B	C	D	E	F
1-			1	Dept	ID	Name	Birthday	Salary	=A1{B7;1}
	1-		2	Admin					=A2{B7}
		1	3	Admin	1	Mike	1968-12-08	8000	=A2{B3;-1:2}
		1	4	Admin	4	Andy	1968-09-19	6000	=A2{B4;-1:2}
	2		5						
	1-		6	R&D					=A6{B9}
		1	7	R&D	2	Jake	1962-02-19	9000	=A6{B7;-1:2}
		1	8	R&D	3	Lucy	1973-08-30	10000	=A6{B8;-1:2}
		1	9	R&D	5	Jim	1965-03-04	4000	=A6{B9;-1:2}
	2		10						

F1 result: [4,2,3,5]. In the range of A1 cell, the peer cells of B7 is B3, B4, B7, B8, and B9. Move a cell downward relative to the B3 cell, then the starting cell is B4, and the end cell is B9 If omitting b. That is, return the cell values of the B4-B9 section.

F2 result: [1,4]. In the range of A2, the peer cells of B7 is B3 and B4. If omitting a/b, then B3 is the starting cell, and B4 is the end cell. The cell values of this section will be returned.

Similarly, the result of F6 is [2,3,5].

F4 result: [1,4]. In the range of A2 cell, the peer cells of B4 are B3 and B4. If the a is -1, then the starting cell is B3; if b is 2, then out of range. Therefore, cell values in the section of B3-B4 are returned.

Similarly, F3 returns [1,4], F7 returns [2,3,5] , F8 returns [2,3,5] , and F9 returns [3,5].

L{A}

Description:

In the range of cell L, get the cell set of peer cells of A

Syntax:

L{A}

Remark:

In the range of cell L, get the cell set composed of peer cells of A . The A can be only used to indicate the peer cells. Therefore, A is not necessarily in the range of cell L.

Parameters:

L Cell

A Cell. Only represent the cell at the same positions

Return value:

Sequence composed of cell value

Example:

0	1	2		A	B	C	D	E	F
1-			1	Dept	ID	Name	Birthday	Salary	=A2{B7}
	1-		2	Admin					=A1{A7}
		1	3	Admin	1	Mike	1968-12-08	8000	=A2{A3}
		1	4	Admin	4	Andy	1968-09-19	6000	=A2{A4}

	2		5						
	1-		6	R&D					=A1{A9}
		1	7	R&D	2	Jake	1962-02-19	9000	=A6{A7}
		1	8	R&D	3	Lucy	1973-08-30	10000	=A6{A8}
		1	9	R&D	5	Jim	1965-03-04	4000	=A6{A9}
	2		10						

F1 result: [1,4], that is, in the range of A2 cell, the peer cells of B7 are B3 and B4. Therefore, [1,4] is returned.

F2 result: ["Admin","Admin","R&D","R&D","R&D"], that is, in the range of A1 cell, the peer cells of A7 are A3, A4, A7, A8, and A9. Therefore, a sequence composed of cell value of cells ["Admin","Admin","R&D","R&D","R&D"] is returned.

F3 result: ["Admin","Admin"], that is, in the range of A2 cell, the peer cells of A3 are A3, and A4. Therefore, ["Admin","Admin"] is returned.

F4 result: ["Admin","Admin"], that is, A is only used to indicate peer cells. Therefore, the same result as F3 is returned.

The L specified in F6 is the same to that in F2, and their peer cells are also the same. Therefore, the same results are returned.

F7 result: ["R&D","R&D","R&D"], that is, in the range of A6 cell, the peer cells of A7 are A7, A8, and A9. Therefore, ["R&D","R&D","R&D"] is returned. In a similar way, the L and peer cells specified by F8 and F9 are the same to that of F7. Therefore, the calculation result is the same to that of F7.

lcm()

Description:

Return the least common multiple

Syntax:

lcm(x_i,…)
lcm(A)

Remark:

Get the least common multiple of members of [x_1, x_2,…]. If there is any member that is equal to or less than 0, than the *lcm* function returns the false value 0

Parameter:

A	Sequence
x_i	The expression, which will be ignored if its value is not a number

Return value:

The least common multiple

Example:

lcm(5,2) **10**

left()

Description:

Get the substring from the left of a string

Syntax:

left(*string*, *n***)**

Remark:

Get the substring from the left of string *string*, the length of which is *n*.

Parameters:

string Get the source string of the substring

n Get the length of the substring

Return value:

Character

Example:

- **left("abcdefg",3)** **"abc"**

Related concepts:

mid()

right()

len()

len()

Description:

Compute the length of string

Syntax:

len(*s***)**

Remark:

Compute the length of string *s*

Parameters:

s String for which you want to compute the length

Return value:

Integer

Example:

- **len("adfg")** **4**

- **len(" abd ")** **5**

A.len()

Description:

Get the length of a sequence

Syntax:

A.**len()**

Remark:

Get the length of the sequence *A*.

Parameters:

A Sequence object

Return value:

Integer

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
2		3	==[B2:F2].len()	==[].len()				

The value of **A3** is: **5**

The value of **B3** is: **0**

Related concepts:

lg()

Description:

Compute the logarithm with 10 as its base

Syntax:

lg(*numberExp*)

Remark:

Compute the logarithm of *numberExp* with 10 as its base

Parameters:

numberExp Data to compute the logarithm with 10 as its base

Return value:

Numeric

Example:

- **lg(54)** **1.7323937598229684**

Related concepts:

ln()

like()

Description:

Judge if a string matches the format string.

Syntax:

like(*stringExp*, *formatExp* **)**

Remark:

Judge if the string *stringExp* matches the format string *formatExp* (**"*"** is to match 0 or multiple characters; **"?"** is to match single character). The escape character can be used to match **"*"**, for example, the result of **like ("abc*123", "abc*")** is true.

Parameters:

stringExp	Expression of character sting
formatExp	Expression of format string

Options:

@c	Indicate not case-sensitive during matching, otherwise, case sensitive by default

Return value:

Boolean

Example:

- **like("abc123", "abc*")** **true**
- **like("abc123", "abc1?3")** **true**
- **like("abc123", "abc*34")** **false**
- **like("abc123", "ABC*")** **false**
- **like@c("abc123", "ABC*")** **true**
- **like ("abc*123", "abc*")** **true**

Linked calculation cell

Description:

Define a linked calculation expression of which the cell is called the linked calculation cell

Syntax:

==*x*

Remark:

Define a calculation expression *x*. Starting with == indicates that other cells are referenced in the *x*. If the value of the referenced cell has changed, then recalculate the *x* linkage.

Parameters:

x	Calculation expression

Return value:

Calculation of expression *x*

Example:

- **==1+1** Cell value is **2**
- **==A1+1** Suppose the cell value of cell **A1** is 2, then the value of 3 will be returned for this cell.

If the cell value of **A1** becomes 3, then this cell will be recalculated, and the value 4 will be returned.

ln()

Description:

Compute the natural logarithm of parameters

Syntax:

ln(*numberExp***)**

Remark:

Compute the natural logarithm of *numberExp*

Parameters:

numberExp Data for which you want to compute the natural logarithm

Return value:

Numeric

Example:

- **ln(54)** **3.9889840465642745**

Related concepts:

lg()

Logic operation

Description:

Perform logical operations on the two boolean expressions

Syntax:

*x***&&***y* Logical AND; If both x and y are true, then the result is true. Otherwise, it is false. As long as value on the left end to the operator is false, the final result will always be false, no matter the value on the right end to the operator is true or false

x||*y* Logical OR; The result is true as long as either x or y is true. Otherwise, it is false. As long as the value on the left end to the operator is true, the final result is always the true, no matter the value on the right end to the operator is true or false.

!*x* Logical NOT, the reverse value of the original value.

Remark:

The operand of logic operation is Boolean. If the operand is not the Boolean, then it will be converted to the Boolean. The result is a Boolean value.

Parameter:

x expression

y expression

Return value:

true/false

Example:

0	1		A	
1-		1	=(2>1)&&(3<4)	true
	1	2	=(2>10)&&(3<4)	false
	2	3	=(2>1)||(3<4)	true
	3	4	=(2>10)||(3<4)	true
	4	5	=(2>11)	false
	5	6	=!(2>11)	true
2		7	=!(12-11)	false

long()

Description:

Convert the value of a string or a number to a 64-bit long integer.

Syntax:

long(*stringExp***)**

long(*numberExp***)**

Remark:

The calculation of *stringExp* must be a string that consist of a long number which is less than or equal to 64 bit. For value more than 64 bits, the result of **long(***stringExp***)** will be imprecise. For value which contains a fractional part, the fractional part will be truncated.

The calculation of *numberExp* must be a long value which is less than or equal to 64 bit. For value more than 64 bits, the result of **long(***numberExp***)** will be imprecise. For value which contains a fractional part, the fractional part will be truncated.

Parameters:

stringExp	The string expression you want to return as a long integer.
numberExp	The number you want to return as a long integer. If the number contains decimal fractions, the fractional part will be truncated.

Return value:

64-bit long

Example:

- **long("1234567")** **1234567**
- **long(1234567.789)** **1234567**

Related concepts:

float()

int()

decimal()

number()

string()

Long integer

Description:

Long integer

Syntax:

1L

Remark:

A long integer is represented by an integer with a tailing capital L.

Example:

- **2345L** **2345**

Related concepts:

Hexadecimal long integer

lookup()

A.lookup()

Description:

Locate all the positions of a member in a sequence, and get the members in these positions of another sequence.

Syntax:

A.**lookup**(*A*$_i$:*x*$_i$,…)

Remark:

Locate all the positions of member x_i in A_i, acquiring the intersection of these positions and return the members in these positions of *A*.

Options:

@a If there are multiple result positions, then return all members. By default, only return the member of the first position of *A*

Parameters:

A_i a sequence

x_i the members in A_i

A the target sequence

Return value:

The members in those result positions of *A*

Example:

- The search after the alignment of main table and sub table

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	99
	1	4	Lily	78	99	80	89	55
2		5		==\{A2\}.lookup@a(\{C2\}:99)		==\{A2\}.lookup(\{D2\}:80)		

B5 result: ["Charles","Lily"]

D5 result: ["Charles"]

loop()

A.loop()

Description:

Iterative loop of a sequence

Syntax:

A.loop(*x;a;c*)

Remark:

Loop the sequence A, ~~ is the result of last running x. The default initial is a. On each run of ~, the result of x will be reassigned to a, and a is null by default. If the result of expression *c* is true, then break off the loop.

Parameters:

a Initial value

x Expression

A Sequence

c An expression that returns true/false

Return value:

Sequence or a certain member value

Example:

0	1		A	B
1-		1	Date	Amount
	1	2	2011-05-10	2456.34
	1	3	2011-05-11	2345.67
	1	4	2011-05-12	855.56
	1	5	2011-05-13	756.87
	1	6	2011-05-14	3456.56
2		7	==\{B2\}.loop@s(~*3;)	Each Amount multiplied by 3
3		8	==\{B2\}.loop(~~*3;2)	
4		9	==\{B2\}.loop(~~*3;2;~>3000)	

A7 result: [7369.02,7037.01,2566.68,2270.61,10369.68]

A8 result: [6,18,54,162,486]

A 9 result: [6,18,54,162]

Related concepts:

A.loops()

loops()

A.loops()

Description:

Perform the cyclic iteration over RSeq and return the result of the last running of *x*

Syntax:

A. **loops(***x;a;c***)**

Remarks:

Operate the RSeq *A* in loops, and the ~~ represents the result of last operation *x*. By default, the initial value is *a*. The ~~ represents the result of *x* will be re-assigned to *a* after each run. The *a* is blank by default. Lastly, return the last computational result of *x*. If the result of expression *c* is true, then break off the loop.

Parameters:

a	Initial value
x	Formula
A	Sequence/RSeq
c	An expression that returns true/false

Return value:

Sequence or value of a certain member

Example:

0	1		A	B
1-		1	Date	Amount
	1	2	2011-05-10	2456.34
	1	3	2011-05-11	2345.67
	1	4	2011-05-12	855.56
	1	5	2011-05-13	756.87
	1	6	2011-05-14	3456.56
2		7	==[B2].loops(~~+~;)	Calculation of cumulative amounts
3		8	==[B2].loops(~~*2;3)	
4		9	==[B2].loops(~~*2;3;~>3000)	

The value of A7 is：9871.0

The value of A8 is：96

The value of A9 is：48

Related concepts:

A.loop()

lower()

Description:

Convert all characters to lower case

Syntax:

lower(*s***)**

Remark:

Convert all characters to lower case

Parameters:

s	Strings you want to convert to the lower case

Return value:

Character

Example:

- **lower("ABCdef")** **"abcdef"**
- **lower("defABC")** **"defabc"**

Related concepts:

upper()

m()

A.m()

Description:

Get members at specified positions.

Syntax:

A.**m**(*i*) -*n*<=*i*<=*n* and *i* is not equal to *0*; For 1<=*i*<=*n*, it indicates to get the ith member; For -*n*<=*i*<=-1, it indicates to get the ith member from the last.

A.**m**(*P*) *P* is the *n* integer sequence whose length is *m*, the member values of which should be larger than -*n* and less than *n*, but not equal to **0**.

Remark:

A is an *n* sequence. Get the members at specified positions *i* or *P* from *A*, which is generally used to get the sequence members reversely.

Parameters:

A a sequence expression

i an integer

P the *n* integer sequence whose length is m (the members are larger than or equal to –*n*, or less than or equal to *n*, but do not equal to 0)

Options:

@r Turn back the position exceeding the boundary of *A*, that is, to set *i*=**if(***i*%*n*==**0,***n*,*i*%*n***)**, where *n* is the length of *A*.

@0 The position exceeding the boundary of *A* will be ignored.

Return value:

A member or a sequence composed of the members at the specified positions in sequence *A*

Example:

0	1		A	B
1-		1	Student	PE Score
	1	2	Aaron	87
	1	3	Charles	90
	1	4	David	75
	1	5	Mary	93
	1	6	Vincent	75
	1	7	Lucy	65

2		8	==\{B2\}.m(2)	==\{B2\}.m(-2)
3		9	==\{B2\}.m([2,3])	==\{B2\}.m([-2,-3])
4		10	==\{B2\}.m@0([5,35])	==\{B2\}.m@r([5,35])

The value of **A8** is: **90**

The value of **B8** is:**75**

The value of **A9** is: **[90,75]**

The value of **B9** is: **[75,93]**

The value of **A10** is: **[75]**

The value of **B10** is: **[75,75]**

Related concepts:

A.p()

max()

A.max()

Description:

Compute the maximum value of all the non-null members in a sequence.

Syntax:

A.**max()** Equivalent to **max**(x_1,…,x_n)

Remark:

Compute the maximum value of all the members in the sequence *A*. Please note that this function cannot be used for a sequence that is composed of the members with different data types.

Parameters:

A A sequence

Return value:

The maximum value of all the members in the sequence *A*

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	91
	1	4	David	75	92	89	96	84
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8	Highest score of each subject	==\{B2\}.max()	==\{C2\}.max ()	==\{D2\}.max()	==\{E2\}.max()	==\{F2\}.max()
3		9		==max(87,null,75,93,75,65,50)				

B8-F8 results are 93,99,98,96, and 98, respectively

B9 result: 93

Related concepts:

A.sum()

A.avg()

A.min()

A.count()

A.variance()

A.max(x)

Description:

Compute x with each member of the sequence and then compute the maximum value of the members of the new sequence

Syntax:

A.**max**(*x*) Equivalent to *A*.(*x*).**max**()

Remark:

Compute x on sequence A by loop and return the maximum value of members of the results. Please note that this function cannot be used for a sequence that is composed of the members with different data types.

Parameters:

A A sequence

x an expression, "**~**" in which is used to reference the current member.

Return value:

The maximum value of members of the result got by performing computation on members in sequence *A*

Example:

0	1		A	B	C	D	E	F	G	H
1-		1	Student	PE	Math	English	History	Geography	MAX	MAX
	1	2	Aaron	87	80	98	80	98	==[B2:F2].max(~)	==[B2:F2].(~+10).max()
	1	3	Charles	90	99	80	76	91	==[B3:F3].max(~)	==[B3:F3].(~+10).max()
	1	4	David	75	92	89	96	84	==[B4:F4].max(~)	==[B4:F4].(~+10).max()
	1	5	Mary	93	78	81	92	76	==[B5:F5].max(~)	==[B5:F5].(~+10).max()
	1	6	Vincent	75	90	88	92	97	==[B6:F6].max(~)	==[B6:F6].(~+10).max()
	1	7	Lucy	65	71	89	69	92	==[B7:F7].max(~)	==[B7:F7].(~+10).max()
	1	8	Lily	aaa	71	89	69	92	==[B8:F8].max(~)	==[B8:F8].(~+10).max()
2		9								

G2-G8 results: 98,99,96,93,97,92, Error message is displayed because the members are of different data types

H2-H8 results: 108,109,106,103,107,102, Error message is displayed because the members are of different data types

Related concepts:

A.max()

maxif()

A.maxif()

Description:

Locate all the positions of a member in a sequence, and get the maximum of the members in these positions of another sequence.

Syntax:

A.**maxif(***A*$_i$:*x*$_i$,…**)**

Remark:

Locate all the positions of member *x*$_i$ in *A*$_i$, acquiring the intersection of these positions and return the maximum value of the members in these positions of *A*

Parameters:

A$_i$ a sequence

x$_i$ the members in *A*$_i$

A the target sequence

Return value:

The maximum value of the members in those result positions of *A*

Example:

0	1		A	B	C	D
1-		1	Class	Name	Subject	Score
	1	2	class one	Aaron	PE	80
	1	3	class one	Bill	PE	89
	1	4	class one	Chris	Math	98
	1	5	class two	Jack	PE	78
	1	6	class two	Chris	PE	90
	1	7	class two	Jack	Math	93
	1	8	class two	Aaron	Math	85
	1	9	class one	Bill	Math	89
2		10	={D2}.maxif({C2}:"PE")			
3		11	={D2}.maxif({C2}:"PE",{A2}:"class one")			

A10 result: 90

A11 result: 89

Related concepts:

A.countif(Ai:xi,…)

A.avgif(Ai:xi,…)

A.minif(Ai:xi,…)

A.sumif(Ai:xi,…)

maxp()

A.maxp()

Description:

Pick out the maximum member of a sequence.

Syntax:

A.**maxp(** *x***)**

Remark:

Compute the expression *x* against each member of the sequence *A* and return the member which makes the value of the expression x maximum

Options:

@1 Return the first member that fulfills the conditions.

@a Return all the members that fulfill the conditions. By default, it is **@1.**

@z Search the members from back to front

Parameters:

A A sequence

x The expression to be calculated

Return value:

The member which makes the value of the expression x maximum

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	2	3		==[B2:F2].maxp(~*~)	==[B2:F2].maxp@a(~*~)	==[B2:F2].maxp@z(~*~)		
	3	4		==[B2:F2].maxp @az(~*~)				

result:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1-	2	Aaron	87	80	98	80	98
	2	3		98	[98,98]	98		
	3	4		[98,98]				

Related concepts:

A.pmax()

A.minp()

merge()

A.merge()

Description:

Merge all sorted A(i) and keep it in order; If x_i is a blank, then use the sequence itself.

Syntax:

A.**merge(***x*$_i$,…**)**

Remark:

Perform merge operation on A(i)|…. If A(i) is sorted for [x_i,…], then the blank x_i indicates it is the sequence itself.

Parameter:

A	Sequence
x$_i$	Members of *A*(i)

Options:

@u	The member of sequence *A(i)* will be merged as a new sequence in proper order and the duplicate members will be removed.
@i	Return the sequence which is composed of same members of sequence *A*(i)
@d	A new sequence generated by removing the members of *A(2)…A(n)* from the sequence *A(i)* one by one.

Return value:

Sequence in the same order as *A*(i)

Example:

	A	
1	=[[3,2,1],[5,4,2],[2],[3]].merge()	[5,4,3,3,2,2,2,1]
2	=[[3,2,1],[5,4,2],[2],[3]].merge@u()	[5,4,3,2,1]
3	=[[3,2,1],[5,3,2]].merge@i()	[3,2]
4	=[[3,2,1],[5,4,2],[2],[3]].merge@d()	[1]

mid()

Description:

Return the substring of a string

Syntax:

mid(*s*, *start*{, *len*}**)**

Remark:

Return the substring of *s*, from the specified position *start*, the length of which is *len*.

Parameters:

s	Source string from which to get the substring
start	Get the starting position of substring
len	Get the length of substring. By default, the length will be counted from the starting character to the end of the source string

Return value:

String

Example:

- **mid("abcde",1)** **abcde**
- **mid("abcde",1,2)** **ab**

- **mid("abcde",3)** **cde**

Related concepts:

left()

right()

millisecond()

Description:

Get the millisecond from a time

Syntax:

millisecond(*datetimeExp***)**

Remark:

Get the millisecond from the time *datetimeExp*.

Parameters:

datetimeExp Expression whose result is a time or date time

Return value:

Integer

Example:

- **millisecond(datetime("1980-02-27 12:00:02:123 ","yyyy-MM-dd hh:mm:ss:SSS"))** **123**
- **millisecond(now())** Milliseconds of the current time

Related concepts:

year()

month()

day()

hour()

minute()

second()

min()

A.min()

Description:

Compute the minimum value of all the non-null members in a sequence.

Syntax:

A.**min()** Equivalent to **min**(x_1,…,x_n)

Remark:

Compute the minimum value of all the members in sequence *A*. Please note that this function should not be used for a sequence that is composed of the members with different data types.

Parameters:

A A sequence

Return value:

The minimum value of all the members in sequence *A*

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	91
	1	4	David	75	92	89	96	84
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8	Lowest score of each subject	==\{B2\}.min()	==\{C2\}.min()	==\{D2\}.min()	==\{E2\}.min()	==\{F2\}.min()
3		9		==min(87,null,75,93,75,65,50)				

B8-F8 results are 65,71,80,69, and 76, respectively

B9 result: 50

Related concepts:

A.sum()

A.avg()

A.count()

A.max()

A.variance()

A.min(x)

Description:

Compute x with each member of the sequence and then compute the minimum value of the members of the new sequence

Syntax:

A.**min**(*x*) Equivalent to *A*.(*x*).**min**()

Remark:

Compute x on sequence *A* by loop and return the minimum value of members of the resulting sequence. Please note that this function cannot be used for a sequence that is composed of the members with different data types.

Parameters:

A A sequence

x an expression, "**~**" in which is used to reference the current member.

Return value:

The minimum value of all members after computation has been performed on sequence *A*

Example:

0	1		A	B	C	D	E	F	G	H
1-		1	Student	PE	Math	English	History	Geography	MIN	MIN
	1	2	Aaron	87	80	98	80	98	==[B2:F2].min(~)	==[B2:F2].(~+10).min()
	1	3	Charles	90	99	80	76	91	==[B3:F3].min(~)	==[B3:F3].(~+10).min()
	1	4	David	75	92	89	96	84	==[B4:F4].min(~)	==[B4:F4].(~+10).min()
	1	5	Mary	93	78	81	92	76	==[B5:F5].min(~)	==[B5:F5].(~+10).min()
	1	6	Vincent	75	90	88	92	97	==[B6:F6].min(~)	==[B6:F6].(~+10).min()
	1	7	Lucy	65	71	89	69	92	==[B7:F7].min(~)	==[B7:F7].(~+10).min()
	1	8	Lily	aaa	71	89	69	92	==[B8:F8].min(~)	==[B8:F8].(~+10).min()
2		9								

G2-G8 results: 80,76,75,76,75,65, Error message is displayed because the members are of different data types

H2-H8 results: 90,86,85,86,85,75, Error message is displayed because the members are of different data types

Related concepts:

A.min()

minif()

A.minif()

Description:

Locate all the positions of a member in a sequence, and get the minimum of the members in these positions of another sequence.

Syntax:

A.**minif(**A_i:x_i, ...**)**

Remark:

Locate all the positions of member x_i in A_i, acquiring the intersection of these positions and return the minimum value of the members in these positions of *A*

Parameters:

A_i a sequence

x_i the members in A_i

A the target sequence

Return value:

The minimum value of the members in those result positions of *A*

Example:

0	1		A	B	C	D
1-		1	Class	Name	Subject	Score
	1	2	class one	Aaron	PE	80
	1	3	class one	Bill	PE	89

	1	4	class one	Chris	Math	98
	1	5	class two	Jack	PE	78
	1	6	class two	Chris	PE	90
	1	7	class two	Jack	Math	93
	1	8	class two	Aaron	Math	85
	1	9	class one	Bill	Math	89
2		10	={D2}.minif({C2):"PE")			
3		11	={D2}.minif({C2):"PE",{A2}:"class one")			

A10 result: 78

A11 result: 80

Related concepts:

A.countif(A_i:x_i,…)

A.avgif(A_i:x_i,…)

A.sumif(A_i:x_i,…)

A.maxif(A_i:x_i,…)

minp()

A.minp()

Description:

Pick out the minimum member of a sequence.

Syntax:

A.**minp(** *x***)**

Remark:

Compute the expression *x* against each member of the sequence *A* and return the member which makes the value of the expression x minimum

Options:

@1 Return the first member that fulfills the conditions.

@a Return all the members that fulfill the conditions. By default, it is **@1.**

@z Search the members from back to front

Parameters:

A A sequence

x the expression to be calculated

Return value:

The member which makes the value of the expression x minimum

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	2	3		==[B2:F2].minp(~*~)	==[B2:F2].minp@a(~*~)	==[B2:F2].minp@z(~*~)		

	3	4		==[B2:F2].minp@az(~*~)			

result:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1-	2	Aaron	87	80	98	80	98
	2	3		80	[80,80]	80		
	3	4		[80,80]				

Related concepts:

A.pmin()

A.maxp()

minute()

Description:

Get the minute from a time

Syntax:

minute(*datetimeExp***)**

Remark:

Get the minute from the specified time *datetimeExp*.

Parameters:

datetimeExp Expression whose result is a time or date time

Return value:

Integer

Example:

- **minute(datetime("19800227","yyyyMMdd"))** **0**
- **minute("1972-11-08 10:20:30")** **20**
- **minute(datetime("2006-01-15 13:20:30"))** **20**

Related concepts:

year()

month()

day()

hour()

second()

millisecond()

month()

Description:

Get the month from a date

Syntax:

month(*dateExp***)**

Remark:

Get the month from the date *dateExp*.

Parameters:

dateExp Expression whose result is a date or date time

Return value:

Integer

Example:

- **month(datetime("19800227","yyyyMMdd"))** **2**
- **month("1972-11-08 10:20:30")** **11**
- **month(datetime("2006-01-15 13:20:30"))** **1**

Related concepts:

year()
day()
hour()
minute()
second()
millisecond()

Mudulos

Description:

Perform Mod operation on two Integer numbers or seek the integer value.

Syntax:

*x***%***y* Get remainder

*x**y* Seek integral value

Remark:

Two integers or long integers division will generate the integer and the remainder parts.

Parameters:

x Integer or Long integer

y Integer or Long integer

Return value:

Integer

Example:

0	1		A	
1-		**1**	**=7%2**	**1**
	1	**2**	**=7\2**	**3**

Related concepts:

cmp()

Multiply sequence

Description:

Generate a new sequence by duplicating members of a sequence.

Syntax:

*A***k* or

*k***A*

Remark:

Generate a new sequence by duplicating the members of the sequence *A* for *k* times

If *k*<=0, then generate an empty sequence.

Parameters:

A a sequence

k an integer

Return value:

The new sequence generated by duplicating the members of the sequence *A* for *k* times.

Example:

0	1		A	
1-		1	=[1,2,3]*3	[1,2,3, 1,2,3, 1,2,3]
	1	2	=3*[1,2,3]	[1,2,3, 1,2,3, 1,2,3]

Related concepts:

Difference sequence

Sequence Union

Concatenate sequence

Alignment Arithmetic Operation

cmp()

n.*f*()

Description:

Compute a loop function using an integer as the loop variable.

Remark:

Compute the loop function *f* using *n* as the loop variable.

n.(*x*) equals to to(*n*).(*x*)

n.*f*(*x*) equals to to(*n*),*f*(*x*)

Parameters:

n An integer

x An expression

f The function name

Example:

- *n*.(*x*)

0		A
1-	1	=3.(~*2)

A1 result: [2,4,6]

- *n.f(x)*

0		A
1	1	=3.sum(~*2)

A1 result: 12

Related concepts:

not()

Description:

On integers, perform bitwise NOT operation to get the logical negation on each bit

Syntax:

not(*x***)**

Remark:

On integers, perform bitwise NOT operation to get the logical negation on each bit

Parameter:

x The numeric expression for which you want to find the logical negation on each bit

Return value:

An integer

Example:

not(6) **-7**

now()

Description:

Get the current system date time

Syntax:

now()

Remark:

Get the current system datetime measured down to the millisecond

Options:

@d Return the date part only, date type

@t Return the time part only, time type

@m Measure to minute

@s Measure to second

Return value:

Date time

Example:

- **now()** The current system date time, for example: **2010-07-15 16:10:40**
- **now@d()** Current system date, for example:**2010-07-15**
- **now@t()** The current system time, for example:**16:10:40**
- **now@m()** The current system time, for example:**2013-12-09 17:05:00:0**
- **now@s()** Current system date, for example:**2013-12-09 17:05:33:0**

null

Description:

Null value

Syntax:

null The value of a null cell is also the **null**.

Remark:

It can be used directly in the constant cell or expression.

Example:

0	1		A
1		1	
2		2	==if(A1==null,0,1)

Related concepts:

true

false

num()

Description:

Get number of peer cells of *A* in the range of level L

Syntax:

num(*A*, *L*) or

Equivalent to **num** (*current cell)*

*L***##** Equivalent to **num** *(current cell,L)*

Remark:

In the range of *L*, get the number of peer cells of *A*

Parameters:

A Cell

L Cell value. It is only used to represent the level. If omitting L, then it is interpreted as the

current cell and the parent cell of *A* that is at the higher level

Return value:

Sequence number

Example:

0	1	2		A	B	C	D	E	F	G	H
1-			1	Dept	ID	Name	Birthday	Salary		=num(A7)	=##
	1-		2	Admin					=num(A3,A1)	=num(A7)	=##
		1	3	Admin	1	Mike	1968-12-08	8000	=num(A3,A2)	=B2##	=B1##
		1	4	Admin	4	Andy	1968-09-19	6000	=num(A4,A2)	=B2##	=B1##
	2		5								
	1-		6	R&D					=num(A7,A1)	=num(A9)	=##
		1	7	R&D	2	Jake	1962-02-19	9000	=num(A7,A6)	=B6##	=B1##
		1	8	R&D	3	Lucy	1973-08-30	10000	=num(A8,A6)	=B6##	=B1##
		1	9	R&D	5	Jim	1965-03-04	4000	=num(A9,A6)	=B6##	=B1##
	2		10								

F3 result:2. Represent the number of peer cells of A3 in the range of level A2, that is, the 2 members of the Admin department.

F7 result: 3. Represent the number of peer cells of A7 in the range of level A6, that is, the 3 members of the R&D department.

F2 result: 5. Represent the number of peer cells of A3 in the range of level A1, that is, there are altogether 5 members from all departments.

G1 result: 5. If omitting L, then it is equivalent to num(A7,A1)

G2 result: 3. If omitting L, then is equivalent to num(A7,A2)

H1,H2,H6 result: 1,2,2. It represents the num(H1,A1), num(H2,A1), and num(H6,A1) respectively

G3,G7 result: 2, 3. Take G3 for example: B2## is equivalent to num(G3,B2)

H3,H7 result: 5,5. Take H3 for example: B1## is equivalent to num(H3,B1)

number()

Description:

Convert a string to a real number.

Syntax:

number(*stringExp***)**

Remark:

The result of *stringExp* must be a numeric string.

Parameters:

stringExp A string expression, the result of which is a numeric string.

Return value:

32-bit integer, 64-bit integer, or 64-bit floating-point number.

Example:

- **number("123")** **123**

- **number("123f")** **123.0**
- **number("123.45")** **123.45**
- **number("123.456d")** **123.456**

Related concepts:

float()
int()
long()
decimal()
string()

Opposite number

Description:

Opposite number

Syntax:

-a

Remark:

Opposite number. If *a* is the date time and character string, then it can be used to sort in descending. Functions frequently used in sorting in ascending order include *cs*.sortx()、*A*.sort(*x* ;loc)、*A*.psort (*x*)、*A*.top(*x*,…;*n*), and etc. By default, they can only be used to sort in ascending order, though the sign "-" can be used to invert the members to be sorted. And sorting the inverted fields ascendingly means sorting the original fields descendingly.

Parameters:

a variable name , data time or string

Return value:

Real number or sequence

Example:

0	1		A	B	C	D	E
1-		1	EID	NAME	STATE	BIRTHDAY	SALARY
	1	2	1	Rebecca	California	1974-11-20	7000
	1	3	2	Ashley	New York	1980-07-19	11000
	1	4	3	Rachel	New Mexico	1970-12-17	9000
	1	5	4	Emily	Texas	1985-03-07	7000
	1	6	5	Ashley	Texas	1975-05-13	16000
	1	7	6	Matthew	California	1984-07-07	11000
	1	8	7	Alexis	Illinois	1972-08-16	9000
2		9	=arg1=5	==-arg1	==\{B2\}.top(-~;3)	==\{D2\}.top(-~;3)	

B9 result: **-5**

C9 result: **["Rebecca","Rachel","Matthew"],** Sort by birthday descendingly

D9 result: **[1985-03-07,1984-07-07,1980-07-19],** Sort by character string descendingly

or()

Description:

Perform bitwise OR operation on integers

Syntax:

or($x_{i,...}$**)**
or(*A***)**

Remark:

Bitwise OR operation on integers

Parameter:

A Sequence
x_i A numeric expression based on which you perform the bitwise OR operation

Retrun value:

An integer

Example:

or(3,5)	**7**

ord()

Description:

Get the sequence number of A among its peer cells in the range of level *L*

Syntax:

ord(*A***, *L***)** or

Equivalent to **ord**(*current cell*)
*L***#** Equivalent to **ord**(*current cell*,*L*)

Remark:

In the range of level *L*, get the sequence number of *A* among its peer cell. The L can be only used to represent the level. Therefore, the L is not necessarily the direct parent cell of A.

Parameters:

A Cell
L Cell; Only use it to represent the level. If omitting L, then it will be interpreted as the current cell, and the parent cell of *A* at the higher level

Return value:

Sequence number

Example:

0	1	2		A	B	C	D	E	F	G	H
1-			1	Dept	ID	Name	Birthday	Salary		=ord(A7)	=#
	1-		2	Admin					=ord(A3,A1)	=ord(A7)	=#
		1	3	Admin	1	Mike	1968-12-08	8000	=ord(A3,A2)	=B2#	=B1#
		1	4	Admin	4	Andy	1968-09-19	6000	=ord(A4,A2)	=B2#	=B1#
	2		5								
	1-		6	R&D					=ord(A7,A1)	=ord(A9)	=#
		1	7	R&D	2	Jake	1962-02-19	9000	=ord(A7,A6)	=B6#	=B1#
		1	8	R&D	3	Lucy	1973-08-30	10000	=ord(A8,A6)	=B6#	=B1#
		1	9	R&D	5	Jim	1965-03-04	4000	=ord(A9,A6)	=B6#	=B1#
	2		10								

F3-F4 result: 1,2, which represent the sequence number of A3 and A4 among its peer cells in the range of A2 level. That is, the sequence number of A3 and A4 in the group of Admin department. The *L* only represents level. Therefore, **=ord(A3,A2)** and **=ord(A3,A6)** will return the same results.

F7-F9 result: 1,2,3, which represent the respective sequence numbers of A7, A8, and A9 among its peer cells in the range of A6 level. That is, the respective sequence numbers of A7, A8, and A9 in the group of R&D department. The *L* only represents the level. Therefore **=ord(A7,A6)** and **=ord(A7,A2)** will return the same result.

F2, F6 result: 1,3, which represent the respective sequence numbers of A3 and A7 among its peer cells in the range of A1 level. That is, the respective sequence numbers of A3 and A7 in all departments.

G1 result: 3; If omitting L, then it equals to ord(A7,A1)

G2 result: 1; If omitting L, then it equals to ord(A7,A2)

H1,H2,H6 result: 1,1,2, representing ord(H1,A1), ord(H2,A1), and ord(H6,A1) respectively

G3-G4,G7-G9 result: 1,2,1,2,and 3. Take G3 for example, B2# equals to ord(G3,B2)

H3-H4,H7-H9 result: 1,2,3,4,and 5. Take H3 for example, B1# equals to ord(H3,B1)

p()

A.p()

Description:

Get sequence numbers of the members at the specified positions.

Syntax:

A.**p**(*i*) -*n*<=*i*<=*n* and *i* is not equal to **0**; For **1**<=*i*<=*n*, it indicates to get the sequence number of the i[th] member; For -*n*<=*i*<=-1, it indicates to get the sequence number of the i[th] member from the last.

A.**p**(*P*) *P* is the *n* integer sequence whose length is *m*, the member values of which should be larger than -*n* and less than *n*, but not equal to **0**.

Remark:

A is an *n* sequence. Get sequence numbers of the members at the specified positions *i* or *P*. This is

generally used to get the sequence numbers of the members reversely.

Options:

@r Turn back the position exceeding the boundary of *A*, that is, to set *i*=**if(***i*%*n*==**0,***n*,*i*%*n***)**, where *n* is the length of *A*.

@0 The position exceeding the boundary of *A* will be ignored.

Parameters:

A a sequence object whose length is *n*

i an integer

P the *n* integer sequence whose length is *m*

Return value:

An integer or an integer sequence of the sequence number of members at the specified positions in the sequence A

Example:

0	1		A	B
1-		1	Student	PE Score
	1	2	Aaron	87
	1	3	Charles	90
	1	4	David	75
	1	5	Mary	93
	1	6	Vincent	75
	1	7	Lucy	65
2		8	==\{B2\}.p(2)	==\{B2\}.p(-2)
3		9	==\{B2\}.p([2,3])	==\{B2\}.p([-2,-3])
4		10	==\{B2\}.p@0([5,35])	==\{B2\}.p@r([5,35])

The value of **A8** is: **2**

The value of **B8** is:**5**

The value of **A9** is: **[2,3]**

The value of **B9** is: **[5,4]**

The value of **A10** is: **[5]**

The value of **B10** is: **[5,5]**

Related concepts:

A.m()

pad()

Description:

Pad another character string ahead of the character string until reaching the specified length.

Syntax:

pad(*s*,*c*,*l*)

Remark:

Pad the character string c ahead of the character string s until the total length of the first character

string is *l*.

Parameters:

s Character string expression

c Character string expression

l Character string whose result is the numeric value

Options:

@r Pad another character string on the right of the character string

Return value:

Character string

Example:

- **pad("Soth","Miss",10)** The return value is **" MissMiSoth "**
- **pad@r("Soth","er",8)** The return value is **" Sotherer "**

parse()

Description:

Parse a string into the corresponding data type

Syntax:

parse(*s*)

Remark:

Analyze the string *s* and parse it into corresponding data type

Parameters:

s String

Options:

@e Remove quotes of the string before starting an escape sequence including the escape character and unicode. By default the string will not be escaped.

Return value:

Data of different data types

Example:

0	1		A	B	C
1-		1	="10:20:30"	\u4e2d\u56fd	a\tb
	1	2	=parse(A1)	=parse@e(B1)	=parse@e(C1)

A2 result:**10:20:30**. The string is parsed into time

B2 result:**中国**. The string is automatically parsed into the corresponding Unicode characters

C2 result: **a** **b**. Escape characters have been handled automatically

pdate()

Description:

Get the first and the last days of the week/month/quarter to which a date belongs

Syntax:

pdate (*dateExp***)**

Remark:

Get the first and the last days of the week/month/quarter to which the date *dateExp* belongs

Parameters:

dateExp Expression whose result is a date or date time

Options:

@w Get the Sunday of the week to which the specified date belongs

@we Get the Saturday of the week to which the specified date belongs

@m Get the beginning day of the month to which the specified date belongs

@me Get the last day of the month to which the specified date belongs

@q Get the beginning day of the quarter to which the specified date belongs

@qe Get the last day of the quarter to which the specified date belongs

The default is to get the Sunday of the week to which the specified date belongs

Return value:

Date time type

Example:

- **pdate@w(datetime("19800227","yyyyMMdd"))** **1980-02-24**
- **pdate@we (datetime("19800227","yyyyMMdd"))** **1980-03-01**
- **pdate@m(datetime("19800227","yyyyMMdd"))** **1980-02-01**
- **pdate@me(datetime("19800227","yyyyMMdd"))** **1980-02-29**
- **pdate@q(datetime("19800227","yyyyMMdd"))** **1980-01-01**
- **pdate@qe(datetime ("19800227","yyyyMMdd"))** **1980-03-31**

periods()

Description:

Generate a date/time sequence by specified interval.

Syntax:

periods(*s*,*e*,*i***)**

Remark:

Generate a new sequence composed of date times from *s* to *e* including start and end points at *i*.

Options:

@y *i* is in years

@q *i* is in quarters

@m *i* is in months

@t *i* is in ten-days

@s *i* is in seconds

@x exclusive of end point

@o Not be adjusted. By default, the result will be adjusted to the original start point of the time unit and adjustment must be done in case of @t.

Parameters:

s a date time variable

e a date time variable

i an integer for indicating the interval; its unit is day and its value is 1 by default

Return value:

The new sequence composed of date times

Example:

0	1		A	B	C
1-		1	1988-01-29 12:54:00	2011-08-30 12:00:00	
	1	2	==periods@y(A1,now(),1)	==periods@y@o(A1,now(),1)	
2-		3			
	1	4	==periods@q(B1,now(),1)	==periods@m(A1,now(),1)	==periods@s(A1,"1988-01-30 02:30:00",3600)
3-		5	==pdate@m(B1)		==after(A5,6-day@w(A5))
	1	6	==periods@x(C5,B1,7)	=periods@t(B1,now(),1)	
4		7	==A6(2)	==A6.m(-1)	==A6.len()

Result:

A2 Set year as the interval unit;

B2 Not adjusted, it is adjusted to the original point of the time unit by default, and must be adjusted when **@t**;

A4 Set quarter as the interval unit;

B4 Set month as the interval unit;

C4 Set second as the interval unit;

A5 The start date of the month;

C5 Get the first Friday;

A6 Get the Friday sequence;

B6 Set ten-days as the interval unit;

A7 Get the second Friday;

B7 Get the last Friday;

C7 How many Fridays;

permut()

Description:

Return the number of permutations

Syntax:

permut(*n*,*k*)

Remark:

The number of ways of rearranging the *k* elements picked from a set of *n* different objects

Parameters:

n An integer that is the number of the objects

k An integer that is the number of each way of permutation

Retrun value:

The number of permutations

Example:

permut(5,4) **120**

pgall()

Description:

Get the total number of pages

Syntax:

pgall()

Remark:

Get the total number of pages

Return value:

Integer, pages

pgcell()

Description:

Get a sequence composed of values of all peer cells of *C* on the current page

Syntax:

pgcell(*C***)**

Remark:

Get a sequence composed of values of all peer cells of *C* on the current page

Parameters:

C Cell

Return value:

A sequence composed of cell value

pgno()

Description:

Get the page number of the current page

Syntax:

pgno()

Remark:

Get the page number of the current page

Return value:

Integer. Page number

pi()

Description:

Compute the circumference ratio and its multiples

Syntax:

pi(*numberExp***)**

Remark:

Compute the circumference ratio and its multiples. The value of *numberExp* is 1 by default.

Parameters:

numberExp Multiples. If omitting this parameter, then return the circumference ratio

Return value:

Circumference ratio and its multiples

Example:

- **pi()** **3.141592653589793**
- **pi(2)** **6.283185307179586**

pmax()

A.pmax()

Description:

Get the position of the maximum member of a sequence.

Syntax:

*A***.pmax(** *x* **,**{*k*}**)**

Remark:

Compute the expression *x* against each member of sequence *A* and return the sequence number of the member whose calculation is the maximum one. The function may be used to locate the position of the maximum value in a sequence.

Options:

@a Return a sequence composed of sequence numbers of all the members that fulfill the rules.

So the result is an *n* integer sequence.

@z Search for the members from back to front from position *k*, and from the last position by default.

Parameters:

A a sequence

x an expression, "**~**" in which is used to reference the current member.

k start the searching from the k^{th} member, and it is **1** by default

Return value:

A sequence number or a sequence composed of sequence numbers

Example:

0	1		A	B	C
1-		1	Id	Name	Math
	1	2	1	Aaron	87
	1	3	2	Bill	100
	1	4	3	Chris	59
	1	5	4	Jack	78
	1	6	5	Lily	65
	1	7	6	Peter	99
	1	8	7	Leon	59
	1	9	8	Anne	100
2		10	==\{C2\}.pmax(~)	==\{C2\}.pmax@a(~)	==\{C2\}.pmax@z(~,3)
3		11	==\{C2\}.pmax@z(~)	==\{C2\}.pmax@az(~)	==\{C2\}.pmax(~,4)

A10 result: 2

B10 result: [2,8]

C10 result: 2

A11 result: 8

B11 result: [8,2]

C11 result: 8

Related concepts:

A.maxp()

A.pmin()

pmin()

A.pmin()

Description:

Get the position of the minimum member of a sequence.

Syntax:

A.**pmin(** *x* {,*k*} **)**

Remark:

Compute the expression *x* against each member of sequence *A* and return the sequence number of the member whose calculation is the minimum one. The function may be used to locate the position of the minimum value in a sequence.

Options:

@a Return a sequence composed of sequence numbers of all the members that fulfill the rules. So the result is an *n* integer sequence.

@z Search for the members from back to front from position *k*, and from the last position by default.

Parameters:

A a sequence

x an expression, "**~**" in which is used to reference the current member.

k start the searching from the k^{th} member, and it is **1** by default

Return value:

A sequence number or a sequence composed of sequence numbers

Example:

0	1		A	B	C
1-		1	Id	Name	Math
	1	2	1	Aaron	87
	1	3	2	Bill	100
	1	4	3	Chris	59
	1	5	4	Jack	78
	1	6	5	Lily	65
	1	7	6	Peter	99
	1	8	7	Leon	59
	1	9	8	Anne	100
2		10	==\{C2\}.pmin(~)	==\{C2\}.pmin@a(~)	==\{C2\}.pmin@z(~,3)
3		11	==\{C2\}.pmin@z(~)	==\{C2\}.pmin@az(~)	==\{C2\}.pmin(~,4)

A10 result: 3

B10 result: [3,7]

C10 result: 3

A11 result: 7

B11 result: [7,3]

C11 result: 7

Related concepts:

A.minp()

A.pmax()

pos()

pos()

Description:

Search the position of a substring in a parent string, and return null if not found

Syntax:

pos(s_1, s_2{, *begin*}**)**

Remark:

Search the position of the substring s_2 in the parent string s_1 from the beginning position *begin*, and return null if not found

Parameters:

s_1	Parent string in which you want to search the substring
s_2	Substring to be searched
begin	The starting character to be searched, and the default is 1

Return value:

Integer

Options:

@z	Search forward starting from the *begin*, and the search will be started from back to forth by default.

Example:

- **pos("abcdef","def")** **3**
- **pos("abcdefdef","def",5)** **6**
- **pos("abcdef","defa")** **null**
- **pos@z("abcdeffdef","def",7)** **4**

A.pos()

Description:

Get the position of a member in a sequence.

Syntax:

A.**pos(***x*{,*k*}**)**

Remark:

Locate the position of a member *x* in sequence *A*, and *x* may appear in *A* repeatedly. The return value is determined by the options. If not found, then return null.

Parameters:

A	a sequence
x	a member
k	start the searching from the k^{th} member, and it is **1** by default

Options:

@b *A* is a sorted sequence by default, so dichotomizing search will be used. Increasing and decreasing are all applicable.

@a Return a sequence composed of sequence numbers of all the members that fulfill the rules. So the result is an n integer sequence.

@z Search for the members from back to front from position k, and from the last position by default.

@s Members of *A* are in order. With the binary search, return the position of *x* if *x* is a member of *A*; otherwise, return a number opposite to the sequence number at which position the *x* can be inserted orderly.

@p If *x* is a sequence, then treat it as an single value. In this case, *A* is a sequence composed of sequences.

@n If no sequence member is found, return the length of A plus 1. This option is mutual exclusive to @a.

Return value:

A sequence number or a sequence composed of sequence numbers

Example:

0	1		A	B	C
1-		1	Id	Name	Score
	1	2	1	Aaron	Excellent
	1	3	2	Bill	Good
	1	4	3	Chris	Pass
	1	5	4	Jack	Excellent
	1	6	5	Lily	Good
	1	7	6	Peter	Excellent
	1	8	7	Leon	Good
	1	9	8	Anne	Pass
2		10	==\{C2\}.pos("Excellent")	==\{B2\}(A10)	==\{A2\}.pos@s(11)
3		11	==\{C2\}.pos@a("Excellent ")	==\{C2\}.pos@az("Excellent")	==\{B2\}.sort(:-1).pos@b("Bill")
4		12	==\{C2\}.pos("Excellent",4)	==\{C2\}.pos@z("Excellent ",2)	==[[2,3],\{A2\}].pos@p([2,3])
5		13	==\{D2\}.pos@n("No Pass")		

A10 result: 1

B10 result: "Aaron". Locate the name of student whose score is “Excellent” according to the sequence number of member retrieved in A10

A11 result: [1,4,6]

B11 result: [6,4,1]

A12 result: 4

B12 result: 1

C10 result: -9

C11 result: 6

C12 result: 1

A13 result: 9

Note:

If *A* is not a sorted sequence, then options **@b and @s** should not be used, or it may bring about the incorrect result. For example C12: ==[B5:B8].pos@b("Leon"), and return the result null

Related concepts:

A.pos(x)

A.psort()

A.pos(x)

Description:

Get the position of a sequence member in another sequence..

Syntax:

A.**pos**(*x*)

Remark:

x is a sequence, return ISeq *p* to make *A*(*p*)==*x*. If not found, then return null.

Parameters:

A a sequence

x a sequence

Options:

@i Return single ascending ISeq *p* to make *A*(*p*)==*x*

@c Return the position in which the sequence x firstly appears in A. By doing so, seek the position of sub sequence x in the sequence A. If x is not the sub sequence of A, then return null.

@b *A* is a sorted sequence by default, so dichotomizing search will be used. Increasing and decreasing are all applicable

Return value:

The unique Ascending integer sequence *p* which makes *A*(*p*)**==***x*

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	81
	1	4	David	75	92	89	96	87
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8	==[B2].pos([93,75,65])	==[B2].pos@i([93,75,65])	==[B2].sort(:-1).pos@b([B4:B6])	==[B2].pos@c([75,93])	==[B2].pos([71,93])	

A8 result: [4,3,6]

B8 result: [4,5,6]

C8 result: [5,1,5]

D8 result: [3]

E8 result: null

Related concepts:

A.pos()

A.psort()

power()

Description:

Compute the powers of a numeric value

Syntax:

power(*x*, *n***)**

Remark:

Compute the *n* powers of *x*

Parameters:

x Base

n Power

Return value:

Numeric

Example:

- **power(2,4)** **16.0**

Related concepts:

exp(n)

product()

Description:

Get the product

Syntax:

product(*x*$_i$,…**)**

product(*A***)**

Remark:

Get the result of multiplying members of [x_1, x_2,…]. Any member that is not a number will be ignored.

Parameter:

A Sequence

x_i An expression

Example:

- =product(17,3,5) 255
- =product(4,"b",12,2) 96

pseg()

A.pseg(*x*)

Description:

Return the position of a member in a sequence

Syntax:

A.**pseg(***x***)**

Remark:

Return the position of *x* in sequence *A*, which must be an ordered one. If *x* does not exist in *A*, return the position where *x* can be inserted by the order.

Parameters:

A A sequence

x An expression

Return value:

The ranking of member *x*

Example:

0	1		A	B
1-		1	Student	PE
	1	2	Aaron	93
	1	3	Charles	90
	1	4	David	85
	1	5	Mary	80
	1	6	Vincent	75
	1	7	Lucy	65
2		8	==[B2].pseg(70)	==[B2].pseg(85)

A8 result: [6]

B8 result: [3]

Related concepts:

A.ranki(y,x)

A.ranki(y)

pselect()

A.pselect()

Description:

Get the positions of the selected members from a sequence.

Syntax:

A.**pselect(** *x* {,*k*}**)**

Remark:

Return the sequence number of a member that fulfils condition *x*, and the return value is subject to the options. If not found, then return the empty sequence or null.

Options:

@a Return a sequence composed of sequence numbers of all the members that fulfill the rules. So the result is an *n* integer sequence.

@z Search for the members from back to front from position *k*, and from the last position by default.

@b *A* is a sorted sequence by default, so dichotomizing search will be used. Increasing and decreasing are all applicable. Note: x_i must be totally sorted ascendingly or descendingly. If *A* is not a sorted sequence, then option **@b** should not be used, or it may bring about the incorrect result.

@s The member in *A* is ordered for formula *x*. With the binary search, if none members in *A* can make the formula *x* generate a result of 0, then return a number opposite to the position at which the number meeting the conditions can be inserted.

@n If no sequence member is found, return the length of *A* plus 1. This option is mutual exclusive to @a

Parameters:

A A sequence

x an Boolean expression, which may be null. when using option **@b**, *x* must be an expression whose return value is a number

k start the searching from the k^{th} member, and it is **1** by default

Return value:

The sequence number of a member that fulfils condition *x*. Use option **@a** to return a sequence composed of all the sequence numbers that fulfils condition *x*, and return a sequence composed of the sequence numbers of all the members when *x* is null.

Example:

0	1		A	B	C
1-		1	Id	Name	Math
	1	2	1	Aaron	87
	1	3	2	Bill	100
	1	4	3	Chris	59
	1	5	4	Jack	78
	1	6	5	Lily	65
	1	7	6	Peter	99
	1	8	7	Leon	59
	1	9	8	Anne	100
2		10	==\{C2\}.pselect(~>85)	==\{C2\}.pselect@a(~>85)	==\{C2\}.pselect@z(~>85,3)
3		11	==\{C2\}.pselect@z(~>85)	==\{C2\}.pselect@az(~>85)	==\{C2\}.pselect(~>85,4)
4		12	==\{A2\}.pselect@s(~:9)	==\{C2\}.sort(:-1).pselect@ab(~-100)	==\{A2\}.pselect@n(~:9)

A10 result: 1
B10 result: [1,2,6,8]
C10 result: 2
A11 result: 8
B11 result: [8,6,2,1]
C11 result: 6
A12 result:-9
B12 result: [1,2]
C12 result: 9

Note:

If *A* is not a sorted sequence, and x_i is totally sorted in ascending or descending order then options **@b and @s** should not be used, or it may bring about the incorrect result.

For example, **==[C2].pselect@ab(~-100)**, the return value is 8.

Related concepts:

A.select()

psort()

A.psort()

Description:

Get the positions of the sorted members of a sequence.

Syntax:

A.**psort (** *x* **)**

Remark:

Generate a new sequence composed of the sequence numbers of all the members in sequence *A* in the order of the values of expression *x*.

Parameters:

A an *n* sequence
x the sorting expression

Options:

@i return the rankings of each member in the original order.

Return value:

The new sequence composed of the sequence numbers of all the members in sequence *A* in the order of the values of expression *x*

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	81
	1	4	David	75	92	89	96	87
	1	5	Mary	93	78	81	92	76

	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8		==\{B2\}.psort(~)		==\{B2\}.psort@i(~)		

B8 result: [6,3,5,1,2,4]

D8 result: [4,5,2,6,3,1]

Related concepts:

A.pos()

A.sort()

A.pos(x)

A.swap(p,q)

A.rvs()

ptop()

A.ptop()

Description:

Get the sequence numbers of the top n smallest members in a sequence

Syntax:

A. **ptop**(*n*,*x*,...)

Remark:

x is an expression based on which each member of a sequence is computed. A sequence comprising the sequence numbers of the *n* smallest members in the original ISeq will be returned. *n* must not be omitted. The omission of *x* is equivalent to ~.

Parameter:

A	A sequence
x	Sort expression
n	Integer

Return value:

A sequence composed of sequence numbers of members

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	81
	1	4	David	75	92	89	96	87
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8		==\{B2\}.ptop(3,~)				

B8 result: [6,5,3]

Related concepts:

rand()

Description:

Get a random value between 0-1.0

Syntax:

rand()

Remark:

Get a random value between 0-1.0

Parameters:

None

Return value:

Random values in the range of [0, 1.0]; float type

Example:

- **rand()** Get a random value in the range of **[0, 1.0]**
- **rand()*100** Get a random float value in the range of **[0, 100]**
- **int(rand()*100)** Get a random integer in the range of **[0, 100]**

rands()

Description:

Get a random character string

Syntax:

rands(*s*,*l***)**

Remark:

Generate a character string of length *l* using the characters from s randomly

Parameter:

s Character string

l Integer

Return value:

Character string

Example:

- **=rands("abc",5)** Get a character string of length 5 comprising the character string "**abc**"

rank()

A.rank()

Description:

Compute the ranking of each member in a sequence.

Syntax:

A.**rank( )**

Remark:

Compute the ranking of each member in sequence *A*, ranking from large to small by default. And generate a new sequence composed of the rankings of members in sequence *A*.

Parameters：

A A sequence

Options:

@z Ranking from small to large. Note: The **"z"** here is in lower case

Return value:

The new integer sequence composed of the rankings of members in sequence *A*

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	91
	1	4	David	75	92	89	96	84
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8	Ranking of each subject	==\{B2\}.rank()	==\{C2\}.rank()	==\{D2\}.rank()	==\{E2\}.rank()	==\{F2\}.rank()
3		9	Backward ranking of each subject	==\{B2\}.rank@z()	==\{C2\}.rank@z()	==\{D2\}.rank@z()	==\{E2\}.rank@z()	==\{F2\}.rank@z()

B8-F8 result: [3,2,4,1,4,6],[4,1,2,5,3,6],[1,6,2,5,4,2],[4,5,1,2,2,6],[1,4,5,6,2,3]

B9-F9 result: [4,5,2,6,2,1],[3,6,5,2,4,1],[6,1,4,2,3,4],[3,2,6,4,4,1],[6,3,2,1,5,4]

Related concepts:

A.rank(x)

A.rank(x)

Description:

Get the ranking of sequence *A*.(*x*)

Syntax:

A.**rank(***x***)** Equivalent to *A*.**(***x***).rank()**

Remark:

Compute the value of expression x according to each member of sequence *A* and return the ranking of sequence *A*.(*x*)

Options:

@z Rank members in ascending order (in descending order by default). Note: Here "**z**" is in lowercase

@i Remove duplicate members from sequence *A*.(*x*) and then compute the ranking

Parameters:

x An expression computed according to sequence *A*

A A sequence

Return value:

The ranking of members of sequence *A*.(*x*)

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	91
	1	4	David	75	92	89	96	84
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8		==\{B2\}.rank(~+10)	==\{C2\}.(~+10).rank()	==\{D2\}.rank@z(~+10)	==\{E2\}.rank@i(~+10)	

B8-E8 results: [3,2,4,1,4,6],[4,1,2,5,3,6],[6,1,4,2,3,4,],[3,4,1,2,2,5]

Related concepts:

A.rank()

ranki()

A.ranki(y)

Description:

Calculate the rankings of the members of a sequence

Syntax:

A.**ranki(** *y***)**

Remark:

Calculate the rankings of *y* in the sequence *A*. By default, it is sorted descendingly, and the return value is the ranking of member *y*.

Option:

@z Rank ascendingly. Please note this is the letter **"z"** in lower case.

Parameters:

A A sequence

y A member of sequence *A*

Return value:

A ISeq composed of rankings of the members of sequence *A*

Example:

0	1		A
1-		1	=[2,1,3,4,8,5]
	1	2	=A1.ranki(6)
2		3	=A1.ranki@z(6)

2, Ranking descendingly

6, Ranking ascendingly

Related concepts:

A.rank()

A.ranki(y,x)

Description:

To obtain the ranking of a value in a sequence.

Syntax:

A.**ranki**(*y*,*x*)

Remark:

Compute the value of expression *x* against each member in *A*, and return the ranking of *y* in *A*.(*x*).

Options:

@z arrange from small to large. Note: The **"z"** here is in lower case

@i remove the duplicate member of *A*.(*x*) first, then obtain the ranking of *y* in *A*.(*x*).

Parameters:

x the calculation expression of *A*

y the member of *A* or a value to be compared with *A*.(*x*)

A a sequence

Return value:

The ranking of member *y*

Example:

0	1		A	B	C	D	E	F	G	H	I
1-		1	Student	PE	Math	English	History	Geography	Total Scores	Ranking of Total Score	Reverse Ranking
	1	2	Aaron	87	80	98	80	98	==[B2:F2].sum()	==A8.ranki(G2)	==A8.ranki@z(G2)
	1	3	Charles	90	99	80	76	91	==[B3:F3].sum()	==A8.ranki(G3)	==A8.ranki@z(G3)
	1	4	David	75	92	89	96	84	==[B4:F4].sum()	==A8.ranki(G4)	==A8.ranki@z(G4)
	1	5	Mary	93	78	81	92	76	==[B5:F5].sum()	==A8.ranki(G5)	==A8.ranki@z(G5)
	1	6	Vincent	75	90	88	92	97	==[B6:F6].sum()	==A8.ranki(G6)	==A8.ranki@z(G6)
	1	7	Lucy	65	71	89	69	92	==[B7:F7].sum()	==A8.ranki(G7)	==A8.ranki@z(G7)

2		8	==\{G2\}						==A8.ranki(400,~+1)		

H2-H7 result: 1,3,3,5,2,6

I2-I7 result: 6,3,3,2,5,1

G8 result: 6, representing 400 ranks the 6th in the [444,437,437,421,443,387].

Related concepts:

A.rank()

regex()

s.regex()

Description:

Match the character string with regular expression

Syntax:

s.**regex(*rs*)**

Remark:

With the regular expression *rs*, match the string *s*, and return an array of section matches. If no match is found, then return null

Options:

@c Case is insensitive

@u Use Unicode

Parameter:

s Character strings

rs Regular expression

Return value:

An array of section matches

Example:

0	1		A	B	C	D
1-		1	4,23,a,test	a, D	W,F	
	1	2	==A1.regex("(\\d),([0-9]*),([a-z]),([a-z]*)")	==B1.regex@c("([a-z]),([a-z])")	==B1.regex("([a-z]),([a-z])")	==C1.regex@u("(\\u0057),(\\u0046)")

A2 result: **["4","23","a","test"]**

B2 result: **["a", "D"]**, Case is insensitive

C2 result: **null,** If no match is found, then return null

D2 result: **["W","F"]**, Use Unicode to match

replace()

Description:

Change the substring of a source string

Syntax:

replace (*src*,*a*,*b*)

Remark:

Change the substring of *src* from *a* to *b*

Parameters:

src	Source string
a	The source substring
b	The target substring

Options:

@q	Quoted characters need not to be replaced

Return value:

String after replacing

Example:

- **replace("abc'abc'def","a","China")** **"Chinabc'Chinabc'def"**
- **replace ("abc'abc'def","a","China")** **"Chinabc'Chinabc'def"**
- **replace@q ("abc'abc'def","a","China")** **"Chinabc'abc'def"**

rgb()

Description:

Convert the red, green, blue, and transparency value to the corresponding color value

Syntax:

rgb(*redIntExp*, *greenIntExp*, *blueIntExp*{, *alphaIntExp*} **)**

Remark:

The value of *redIntExp*, *greenIntExp*, *blueIntExp*, *alphaIntExp* must be between 0-255.

Parameters:

redIntExp	The integer expression to indicate the red, of which the value is between 0-255
greenIntExp	The integer expression to indicate the green, of which the value is between 0-255
blueIntExp	The integer expression to indicate the blue, of which the value is between 0-255
alphaIntExp	The integer expression to indicate the transparency, of which the value is between 0-255 The 0 represents the radical transparent, and 255 indicates totally opaque The other values respectively represent the transparency of various levels and the default is 255

Return value:

The 64 bit long integer

Example:

- **rgb(123,123,123)** **-8684677**
- **rgb(123,123,123,123)** **2071690107**
- **rgb(123,123,123,255)** **-8684677**
- **rgb(123,123,123,0)** **8092539**

right()

Description:

Get the substring from the right of a string

Syntax:

right(*s*, *n***)**

Remark:

Get the substring from the right of string *s*, the length of which is *n.*

Parameters:

s Source string from which to get the substring

n Get the length of substring

Return value:

String

Example:

- **right("abced",2)** **"ed"**

Related concepts:

left()

mid()

round()

Description:

Truncate the data at the specified position, and round off the remaining part

Syntax:

round(*numberExp*, {*nExp*}**)**

Remark:

Truncate the data *numberExp* at the specified position *nExp*, and round off the remaining part

Parameters:

numberExp	Data to be intercepted
nExp	Integer to specify the position at which to intercept
	>0: Move the decimal point to the right for *nExp* places
	<0: Move the decimal point to the left for *nExp* places
	=0: Indicate the current decimal places.

Return value:

Numeric

Example:

- **round(3451251.274,0)** **3451251.0**
- **round(3451251.274,-1)** **3451250.0**
- **round(3451251.274,-2)** **3451300.0**
- **round(3451251.274,1)** **3451251.3**
- **round(3451251.274,2)** **3451251.27**

Related concepts:

ceil()

floor()

row()

Description:

Get the row number of the current row

Syntax:

row()

Remark:

Get the row number of the current row

Return value:

Integer, row number

Example:

0	1	2		A	B	C	D	E	F
1-			1	Dept	ID	Name	Birthday	Salary	=row()
	1-		2	Admin					=row()
		1	3	Admin	1	Mike	1968-12-08	8000	=row()
		1	4	Admin	4	Andy	1968-09-19	6000	=row()
	2		5						=row()
	1-		6	R&D					=row()
		1	7	R&D	2	Jake	1962-02-19	9000	=row()
		1	8	R&D	3	Lucy	1973-08-30	10000	=row()
		1	9	R&D	5	Jim	1965-03-04	4000	=row()
	2		10						=row()

F1-F10 results are 1,2,3,4,5,6,7,8,9, and 10, respectively

run()

A.run()

Description:

Compute expressions against each member in a sequence and return the sequence itself.

Syntax:

A.**run**(*x*)

Remark:

Compute the expressions x_i against each member in *A* and return *A* itself. "**~**" in *x* is used to reference the current member in *A*.

Parameters:

A a sequence

x an expression, "**~**" in which is used to reference the current member.

Return value:

Sequence *A* whose member values may have been modified

Example:

Use "**run**" function to modify the member values

0	A
1	=3.(~*2)
2	=[1,2,3,4,5,6]
3	==A2.run(~=~*~)

A1 result: [2,4,6]

A2 result: [1,4,9,16,25,36], use"~" to reference the current member

Note:

The difference between *A*.(*x*) and *A*.**run**(*x*) :

The aim of *A*.(*x*) is to compute the values of expression *x* and return a sequence that is composed of the values of this expression;

The aim of *A*.**run**(*x*) is to make some changes on *A* through the computation of *x* and thereby return *A* which has been modified

Related concepts:

rvs()

A.rvs()

Description:

Generate a new sequence by reversing the members in a sequence.

Syntax:

A.**rvs()**

Remark:

Generate a new sequence by reversing the members in *A*.

Parameter:

A A sequence

Return value:

The new sequence by reversing the members in *A*

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	81
	1	4	David	75	92	89	96	87
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8		==[B2].rvs()				

B8 result: [65,75,93,75,90,87].

Related concepts:

A.psort()

A.sort()

A.swap(p,q)

s()

Description:

Concatenate parameters into a string where sequences will be splitted

Syntax:

s(x_i,…**)**

Remark:

Concatenate parameters into a string in which quotation marks will not be used

Parameter:

x_i Any value that can be converted to a string. If it is is sequence, it will be broken up

Return value:

A string

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	81
	1	4	David	75	92	89	96	87
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8		==s({A2},{B2})	=s(2,3,"a")			

B8 result: AaronCharlesDavidMaryVincentLucy879075937565

C8 result: 23a

Related concepts:

A.string()

s. array()

second()

Description:

Get the second from a time

Syntax:

second(*datetimeExp***)**

Remark:

Get the second from the time *datetimeExp*.

Parameters:

datetimeExp Expression whose result is a time or date time

Return value:

Integer

Example:

- **second(datetime("19800227","yyyyMMdd"))** **0**
- **second("1972-11-08 10:20:30")** **30**
- **second(datetime("2006-01-15 13:20:45"))** **45**

Related concepts:

year()
month()
day()
hour()
minute()
millisecond()

select()

A.select()

Description:

Pick out members from a sequence which satisfied a condition.

Syntax:

A.**select(** *x* **)**

Remark:

Compute the expression *x* against each member of the sequence *A*, then generate a new sequence composed of those members which make the value of the expression x to be true.

Options:

@1 return the first member that fulfills the conditions. By default, it is **@a**

@z Search the members from back to front

@b The *A* is a sorted sequence by default, so dichotomizing search will be used. Increasing and decreasing are all applicable. Note: If *A* is not a sorted sequence, then option **@b** should not be used, or it may bring about the incorrect result.

@o Do not create a new sequence, but alter the original sequence *A* instead.

Parameters:

A A sequence

x an Boolean expression, which may be null. when using option **@b**, *x* must be an expression whose return value is a number

Return value:

The new sequence composed of those members which make the value of the expression x to be true

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	2	3		==[B2:F2].select(~>85)	==[B2:F2].select@1(~>85)	==[B2:F2] .select@z(~>85)		
	3	4				==[B2:F2] .select@b(~-80)		

result:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1-	2	Aaron	87	80	98	80	98
	2	3		[87,98,98]	87	[98,98,87]		
	3	4				[]		

Note:

If *A* is not a sorted sequence, then option **@b** should not be used, or it may bring about the incorrect result.

For example, D4: ==[B2:F2].select@b(~-80), and return [].

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	2	3	=[B2:F2]	==A3.select(~>85)	==A3.select@o(~>85)			

B3 result: [87,98,98], and A3 will not be changed.

C3 result: [87,98,98], and the original sequence *A*3 will be changed.

Related concepts:

A.pselect()

Sequence Union

Description:

Generate a new sequence by merging two sequences.

Syntax:

*A***&***B*

Remark:

Generate a new sequence by merging the members (or single values) from the two sequences *A* and *B* in proper order. The common members will not appear repeatedly in the new sequence.

Parameters:

A an *n* sequence or a single value; When it is a single value, it is regarded as **[***A***]**

B an *m* sequence or a single value; When it is a single value, it is regarded as **[***B***]**

Return value:

The new sequence after merging the two sequences *A* and *B*

Example:

0	1		A	B
1-		1	Student	English
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	Math
	1	9	Vincent	100
	1	10	Aaron	99
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==A2}(to(3))&A9}(to(3))	

Either math or English score is among the top 3

The value of **A15** is: **["Aaron","Charles","David","Vincent"]**, **"Aaron"** and **"Charles"** will not appear repeatedly.

Related concepts:

Difference sequence

Intersection sequence

Concatenate sequence

Alignment Arithmetic Operation

cmp()

shift()

Description:

Perform shift operation

Syntax:

shift(*x***,***s***)**

Remark:

Shift left if s<0; shift right if s>0

Parameters:

x The expression based on which you perform the shift operation

s An integer

Option:

@s When shifting right, shift a sign into the leftmost position and shif a zero to this position by default

Return value:

An integer

Example:

shift(13,2) **3**

sign()

Description:

To judge whether the parameter is positive, negative or 0

Syntax:

sign(*number***)**

Remark:

If *number* is positive value, return **1**; If it is negative value, return **-1**; If it is 0, return **0**

Parameters:

number Data for which you want to judge whether it is positive or negative

Return value:

Integer

Example:

- **sign(-10)** **-1**
- **sign(30)** **1**
- **sign(0)** **0**

sin()

Description:

Compute the sine value.

Syntax:

sin(*number***)**

Remark:

The parameter *number* is in radians.

Parameters:

number Radians for which you want to compute the sine

Return value:

float type

Example:

- **sin(pi())** **1.2246467991473532E-16**
- **sin(pi(2))** **-2.4492935982947064E-16**
- **sin(pi()/2)** **1.0**

Related concepts:

cos()

tan()

sinh()

Description:

Return the hyperbolic sine

Syntax:

sinh(*number***)**

Remark:

The parameter *number* is any real number

Parameter:

number The real number for which you want to find the hyperbolic sine

Return value:

The hyperbolic sine

Example:

sinh(1) **1.1752011936438014**

sort()

A.sort()

Description:

Generate a new sequence by sorting the members of a sequence.

Syntax:

A.sort(*x*;*loc*)

Remark:

Generate a new sequence by sorting the members of sequence *A* according to the values of *x* and local language *loc*. If omitting *loc*, then compare Unicode value.

Parameters:

A A sequence

x an expression, according to which the members of sequence *A* will be sorted ascendingly.

loc Language name

Values for *loc*:

ja_JP	Japanese	Japan
es_PE	Spanish	Peru
en	English	
ja_JP_JP	Japanese	Japan JP
es_PA	Spanish	Panama
sr_BA	Serbian	Bosnia and Herzegovina
mk	Macedonian	
es_GT	Spanish	Guatemala
ar_AE	Arabic	United Arab Emirates
no_NO	Norwegian	Norway
sq_AL	Albanian	Albania
bg	Bulgarian	
ar_IQ	Arabic	Iraq
ar_YE	Arabic	Yemen
hu	Hungarian	
pt_PT	Portuguese	Portugal
el_CY	Greek	Cyprus
ar_QA	Arabic	Qatar
mk_MK	Macedonian	Macedonia
sv	Swedish	
de_CH	German	Switzerland
en_US	English	United States
fi_FI	Finnish	Finland
is	Icelandic	
cs	Czech	
en_MT	English	Malta
sl_SI	Slovenian	Slovenia
sk_SK	Slovak	Slovakia
it	Italian	
tr_TR	Turkish	Turkey
zh	Chinese	
th	Thai	

ar_SA	Arabic	Saudi Arabia
no	Norwegian	
en_GB	English	United Kingdom
sr_CS	Serbian	Serbia and Montenegro
lt	Lithuanian	
ro	Romanian	
en_NZ	English	New Zealand
no_NO_NY	Norwegian	Norway Nynorsk
lt_LT	Lithuanian	Lithuania
es_NI	Spanish	Nicaragua
nl	Dutch	
ga_IE	Irish	Ireland
fr_BE	French	Belgium
es_ES	Spanish	Spain
ar_LB	Arabic	Lebanon
ko	Korean	
fr_CA	French	Canada
et_EE	Estonian	Estonia
ar_KW	Arabic	Kuwait
sr_RS	Serbian	Serbia
es_US	Spanish	United States
es_MX	Spanish	Mexico
ar_SD	Arabic	Sudan
in_ID	Indonesian	Indonesia
ru	Russian	
lv	Latvian	
es_UY	Spanish	Uruguay
lv_LV	Latvian	Latvia
iw	Hebrew	
pt_BR	Portuguese	Brazil
ar_SY	Arabic	Syria
hr	Croatian	
et	Estonian	
es_DO	Spanish	Dominican Republic
fr_CH	French	Switzerland
hi_IN	Hindi	India
es_VE	Spanish	Venezuela
ar_BH	Arabic	Bahrain
en_PH	English	Philippines
ar_TN	Arabic	Tunisia
fi	Finnish	
de_AT	German	Austria

es	Spanish	
nl_NL	Dutch	Netherlands
es_EC	Spanish	Ecuador
zh_TW	Chinese	Taiwan
ar_JO	Arabic	Jordan
be	Belarusian	
is_IS	Icelandic	Iceland
es_CO	Spanish	Colombia
es_CR	Spanish	Costa Rica
es_CL	Spanish	Chile
ar_EG	Arabic	Egypt
en_ZA	English	South Africa
th_TH	Thai	Thailand
el_GR	Greek	Greece
it_IT	Italian	Italy
ca	Catalan	
hu_HU	Hungarian	Hungary
fr	French	
en_IE	English	Ireland
uk_UA	Ukrainian	Ukraine
pl_PL	Polish	Poland
fr_LU	French	Luxembourg
nl_BE	Dutch	Belgium
en_IN	English	India
ca_ES	Catalan	Spain
ar_MA	Arabic	Morocco
es_BO	Spanish	Bolivia
en_AU	English	Australia
sr	Serbian	
zh_SG	Chinese	Singapore
pt	Portuguese	
uk	Ukrainian	
es_SV	Spanish	El Salvador
ru_RU	Russian	Russia
ko_KR	Korean	South Korea
vi	Vietnamese	
ar_DZ	Arabic	Algeria
vi_VN	Vietnamese	Vietnam
sr_ME	Serbian	Montenegro
sq	Albanian	
ar_LY	Arabic	Libya
ar	Arabic	

zh_CN	Chinese	China
be_BY	Belarusian	Belarus
zh_HK	Chinese	Hong Kong
ja	Japanese	
iw_IL	Hebrew	Israel
bg_BG	Bulgarian	Bulgaria
in	Indonesian	
mt_MT	Maltese	Malta
es_PY	Spanish	Paraguay
sl	Slovenian	
fr_FR	French	France
cs_CZ	Czech	Czech Republic
it_CH	Italian	Switzerland
ro_RO	Romanian	Romania
es_PR	Spanish	Puerto Rico
en_CA	English	Canada
de_DE	German	Germany
ga	Irish	
de_LU	German	Luxembourg
de	German	
es_AR	Spanish	Argentina
sk	Slovak	
ms_MY	Malay	Malaysia
hr_HR	Croatian	Croatia
en_SG	English	Singapore
da	Danish	
mt	Maltese	
pl	Polish	
ar_OM	Arabic	Oman
tr	Turkish	
th_TH_TH	Thai	Thailand TH
el	Greek	
ms	Malay	
sv_SE	Swedish	Sweden
da_DK	Danish	Denmark
es_HN	Spanish	Honduras

Options:

@o Do not create a new sequence, but alter the original sequence *A* instead.

Return value:

The new sorted sequence

Example:

0	1		A	B	C	D	E	F

1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	81
	1	4	David	75	92	89	96	87
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8	={B2}	==A8.sort()	==A8.sort@o()	==[B1:F1].sort(;"en")		

B8 result: [65,75,75,87,90,93], and A8 will not be changed.

C8 result: [65,75,75,87,90,93], and the original sequence *A8* will be changed

D8 result: ["English","Geography","History","Math","PE"], sort accord to the English.

Related concepts:

A.psort()

A.swap(p,q)

A.rvs()

sqrt()

Description:

Compute the square root

Syntax:

sqrt(*number***)**

Remark:

Compute the square root

Parameters:

number Data for which you want to compute the square root

Return value:

Numeric

Example:

- **sqrt(100)** **10.0**
- **sqrt(99)** **9.9498743710662**

step()

A.step()

Description:

Get members from a sequence with a starting position and a step, so as to create a new sequence.

Syntax:

A.**step(***m*,k_i,...**)**

Remark:

Search for the members whose sequence numbers are k_i, k_i+m, k_i+2m,…from *A* to compose a new sequence.

Parameters:

m a positive integer used to specify the span

k_i the starting sequence number, $1<=k_i<=m$

A a sequence whose length is *n*

Return value:

A new sequence whose length is *m*

Example:

0	1		A	B	C	D	E	F	G	H	I
1		1	Student	The 1st exam		The 2nd exam		The 3rd exam		The average score	
2-		2		Math	English	Math	English	Math	English	Math	English
	1	3	Aaron	87	80	98	80	98	80	==[B3:G3].step(2,1,2).avg()	==[B3:G3].step(2,2,2).avg()
	1	4	Charles	90	99	80	76	91	99	==[B4:G4].step(2,1,2).avg()	==[B4:G4].step(2,2,2).avg()
	1	5	David	75	92	89	96	84	80	==[B5:G5].step(2,1,2).avg()	==[B5:G5].step(2,2,2).avg()
	1	6	Mary	93	78	81	92	76	76	==[B6:G6].step(2,1,2).avg()	==[B6:G6].step(2,2,2).avg()
	1	7	Vincent	75	90	88	92	97	84	==[B7:G7].step(2,1,2).avg()	==[B7:G7].step(2,2,2).avg()
	1	8	Lucy	65	71	89	69	92	76	==[B8:G8].step(2,1,2).avg()	==[B8:G8].step(2,2,2).avg()
3		9									

The value of **H3** is: **87.16**

The value of **I3** is: **80.0**

string()

string(expression{, format})

Description:

Convert the object of other type to the string type and format it.

Syntax:

string(*expression*{, *format*})

Remark:

The format string *format* must match the data type of the result of *expression*, or the result of string(*expression*{, *format*}) may be incorrect.

Parameters:

expression The constant object or expression to be converted to string.

format A format string used to format the result of *expression*

Options:

@q Enclosed the string *expression* in double quotes and ignore parameter *format*

@e Escape the undisplayable character. Represent the tab, carriage return, line break in the string *expression* with the escape characters. Add an escape character before the

single quotes, double quotes or an escape character if there is any in the string. Ignore parameter *format*

@u With the use of @e option, if there is large character set in the string *expression*, convert it to Unicode characters

Return value:

String

Example:

0	1		A	B	C	D	E	F	G	H
1-		1					a b		a\b	中国
	1	2	==string(123)	==string(date("2009-02-23")," MMM dd, yyyy")	==string(3456.78,"$#,##0.00")	==string(5/6,"0.00%")	==string@q(E1,"|")	==string@e(E2,"|")	==string@q(G1)	=string@u(H1)

A2 result: **123**

B2 result: **Feb 23, 2009**

C2 result: **$3,456.78**

D2 result: **83.33%**

E2 result: **"a b"**

F2 result: **\"a\tb\"**

G2 result: **"a\\b"**

H2 result: **\u4E2D\u56FD**

Related concepts:

float()
int()
long()
number()
decimal()

A.string(d)

Description:

Join all the members of a sequence with a delimiter.

Syntax:

A.**string(***d***)**

Remark:

Join the members of *A* into a string delimited by *d*, and the subsequence member will be processed.

Options:

@q Add quotation marks when concatenating strings. If this option is omitted, do not use the quotation marks.

Parameters:

A String sequence

d Delimiter and the default is the comma

Return value:

A string after joining

Example:

0		A	B	C
1	1	==1	==["a","b"]	==[2,"c"]
2	2	==[A1:C1].string()	==[A1:C1].string(":")	=[A1:C1].string@q()

A2 result: **1,[a,b],[2,c]**

B2 result: **1:[a:b]:[2:c]**

C2 result: **1,["a","b"],[2,"c"]**

Related concepts:

s. array()

String

Description:

Define a string constant.

Syntax:

"*string*"

Remark:

The expression must be double quoted when using. But the quotation mark is not required if the character string constants are defined directly.

When copying/pasting/inserting/deleting the row, the cell name in "*string*" will not change automatically.

The double quotation mark in "*string*" needs to use the escape character.

Parameters:

string Content of the string. Content can be any character.

Return value:

String constant

Example:

- **"asd"+"sfd"**
- **dfg**

For the character string enclosed in the "", when copying, pasting, adding, or deleting rows, the cell name in the character string will not change automatically.

0	1		A	B
1		1	=="a"	=="b"
2		2	=="c"	=="d"
3		3	=="A2"+"e"	==$[B2]+"f"

The cell name is not changed.

After deleting the first row, the cell changed like this:

0	1		A	B
1		1	=="c"	=="d"
2		2	=="A2"+"e"	==$[B1]+"f"

The double quotation mark in **""** needs to use the escape character.

- **"a\"s"**

Related concepts:

Escape character

String concatenation

String concatenation

Description:

Join two or more strings end-to-end.

Syntax:

x+*y*

Remark:

A string and a numeric value cannot be concatenated.

Parameters:

x A string constant

y A string constant

Return value:

A concatenated string formed by joining *x* and *y*.

Example:

- **"abc"+"def"** **abcdef**
- **"abc"+123** **123**. A string and a numeric value cannot be concatenated.

sum()

A.sum()

Description:

Compute the sum of all the members in a sequence.

Syntax:

A.**sum()** Equivalent to **sum**(x_1,…,x_n)

Remark:

Compute the summary value of members in sequence *A*. Skip those members that are not numerical values.

Parameters:

A A sequence

Return value:

The sum of all the members in *A*

Special description:

The null member is processed as **0**

Example:

0	1		A	B	C	D	E	F	G
1-		1	Student	PE	Math	English	History	Geography	Sum
	1	2	Aaron	87	80	98	80	98	==[B2:F2].sum()
	1	3	Charles	90	99	80	76	91	==[B3:F3].sum()
	1	4	David	75	92	89	96	84	==[B4:F4].sum()
	1	5	Mary	93	78	81	92	76	==[B5:F5].sum()
	1	6	Vincent	75	90	88	92	97	==[B6:F6].sum()
	1	7	Lucy	65	71	89	69	92	==[B7:F7].sum()
	1	8	Lily	aaa	71	89	69	92	==[B8:F8].sum()
2		9		==sum(87,"a",75,93,75,65,50)					

G2-G8results are 443,436,436,420,442, 386, and 321

B9 result: 445

Related concepts:

A.count()

A.avg()

A.min()

A.max()

A.variance()

A.sum(x)

Description:

Compute x with each member of the sequence and compute the summary value of the members of the new sequence

Syntax:

A.**sum**(*x*) Equivalent to *A*.(*x*).**sum**()

Remark:

Compute *x* on sequence *A* by loop and return the summary value of members of the resulting sequence

Parameters:

A A sequence

x An expression, "**~**" in which is used to reference the current member. The data type of the computed result of the expression is numerical value.

Return value:

A numerical value

Special Note:

Take a null value as zero

Example:

0	1		A	B	C	D	E	F	G	H
1-		1	Student	PE	Math	English	History	Geography	SUM	SUM
	1	2	Aaron	87	80	98	80	98	==[B2:F2].sum(~)	==[B2:F2].(~+10).sum()

	1	3	Charles	90	99	80	76	91	==[B3:F3].sum(~)	==[B3:F3].(~+10).sum()
	1	4	David	75	92	89	96	84	==[B4:F4].sum(~)	==[B4:F4].(~+10).sum()
	1	5	Mary	93	78	81	92	76	==[B5:F5].sum(~)	==[B5:F5].(~+10).sum()
	1	6	Vincent	75	90	88	92	97	==[B6:F6].sum(~)	==[B6:F6].(~+10).sum()
	1	7	Lucy	65	71	89	69	92	==[B7:F7].sum(~)	==[B7:F7].(~+10).sum()
	1	8	Lily	aaa	71	89	69	92	==[B8:F8].sum(~)	==[B8:F8].(~+10).sum()
2		9								

G2-G8 results: 443,436,436,420,442,386, 321

H2-H8 results: 493,486,486,470,492,436, 371

Related concepts:

A.sum()

sumif()

A.sumif()

Description:

Locate all the positions of a member in a sequence, and get the sum of the members in these positions of another sequence.

Syntax:

A.**sumif(***A*$_i$:*x*$_i$, ...**)**

Remark:

Locate all the positions of member *x*$_i$ in *A*$_i$, acquiring the intersection of these positions and return the sum of the members in these positions of *A*

Parameters:

A$_i$ a sequence

x$_i$ the members in *A*$_i$

A the target sequence

Return value:

The sum of the members in those result positions of *A*

Example:

0	1		A	B	C	D
1-		1	Class	Name	Subject	Score
	1	2	class one	Aaron	PE	80
	1	3	class one	Bill	PE	89
	1	4	class one	Chris	Math	98
	1	5	class two	Jack	PE	78
	1	6	class two	Chris	PE	90
	1	7	class two	Jack	Math	93
	1	8	class two	Aaron	Math	85
	1	9	class one	Bill	Math	89
2		10	={D2}.sumif({C2}:"PE")			

3		11	={D2}.sumif({C2}:"PE",{A2}:"class one")

A10 result: 337

A11 result: 169

Related concepts:

A.countif(A_i:x_i,…)

A.avgif(A_i:x_i,…)

A.minif(A_i:x_i,…)

A.maxif(A_i:x_i,…)

swap()

A.swap()

Description:

Generate a new sequence by swapping the member positions of two specified intervals of a sequence.

Syntax:

A.**swap**(*p*,*q*)

Remark:

Generate a new sequence by swapping the member positions of two specified intervals in sequence *A*, and the two intervals should not overlap each other.

Parameters:

A A sequence

p an integer sequence interval composed of positive integers, for example **[1,2,3], to (1,3)**

q an integer sequence interval composed of positive integers and does not have intersection with *p*, for example **[4,5,6], to(4,6)**

Return value:

The new sequence after swapping

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	91
	1	4	David	75	92	89	96	84
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8	==[B2:F7]	==A8.swap(to(1,5),to(11,15))	==A8.swap(to(1,6),to(6,15))			

B8 result:

[75,92,89,96,84,90,99,80,76,91,87,80,98,80,98,93,78,81,92,76,75,90,88,92,97,65,71,89,69,92]

C8 reports error: the two intervals must not overlap each other

Note:

The two intervals to be swapped in a sequence should not overlap each other.

Related concepts:

tan()

Description:

Compute the tangent value

Syntax:

tan(*number***)**

Remark:

The parameter *number* is in radians

Parameters:

number Radians for which you want to compute the tangent value

Return value:

float type

Example:

- **tan(pi()/2)** **1.633123935319537E16**
- **tan(pi(2))** **-2.4492935982947064E-16**

Related concepts:

sin()

cos()

tanh()

Description:

Return the hyperbolic tangent

Syntax:

tanh(*number***)**

Remark:

The parameter number is a real number

Parameter:

number The real number for which you want to find the hyperbolic tangent

Example:

tanh(0.5) **0.46211715726000974**

time()

time(*datetimeExp*)

Description:

Get the time part from the datetime value

Syntax:

time(*datetimeExp***)**

Remarks:

Get the time part from *datetimeExp*, accurate to millisecond by default. The format must be consistent with the time format in the designer option. By default, the designer option will not be displayed in millisecond.

Parameters:

datetimeExp datetime

Options:

@m Measure to minute

@s Measure to second

Return value:

Time value

Example:

- **time(now())** **16:28:26400**
- **time@s(now())** **16:28:260**
- **time@m(now())** **16:28:000**

Related concepts:

date()

date(*datetimeExp*)

datetime(*datetimeExp*)

datetime()

time()

time()

Description:

Convert the string or integer to time data

Syntax:

time(*stringExp*{, *format* }) Convert *stringExp* to time data type according to the format specified by *format*. If there is no parameter *format*, format of the string *stringExp* must be in consistent with the time format in the configuration information

time(*h*,*m*,*s*) Convert *h*,*m*,*s* of integer type to time data type

Remark:

Convert the string *stringExp* or integer *h*,*m*,*s* to time data

Parameters:

stringExp	A string
format	A string specifying data format
h	integer
m	integer
s	integer

Return value:

Time data

Example:

- **time("00:00:45")** **00:00:45**
- **time(12,13,00)** **12:13:00**
- **time("00/00/45","hh/mm/ss")** **00:00:45**

Related concepts:

datetime()

date()

date(*datetimeExp*)

datetime(*datetimeExp*)

time(*datetimeExp*)

to()

to()

Description:

Generate an integer sequence.

Syntax:

to(*a*,*b*)	Generate an integer sequence composed of continuous integers between *a* and *b*.
to(*n*)	Generate an integer sequence composed of continuous integers from **1** to *n*.

Remark:

Generate an integer sequence composed of a set of continuous integers from *a* to *b* or from 1 to *n*.

Parameters:

a	the starting integer
b	the ending integer
n	*n*>0

Options:

@s Generate a sequence composed of *b* integers continuously starting from *a*, If *b* is less than 0, it is generated backward one by one in descending order.

Return value:

A continuous integer sequence

Example:

0		A	B	C	D	E	F	G
1	1	==to(3,7)	==to(5,3)	==to(-2,3)	==to(3,-2)	==to@s(3,4)	==to@s(3,-2)	==to(10)

The value of **A1** is: **[3,4,5,6,7]**

The value of **B1** is: **[5,4,3]**

The value of **C1** is: **[-2,-1,0,1,2,3]**

The value of **D1** is: **[3,2,1,0,-1,-2]**

The value of **E1** is: **[3,4,5,6]**

The value of **F1** is: **[3,2]**

The value of **G1** is: **[1,2,3,4,5,6,7,8,9,10]**

Related concepts:

A.to()

A.to()

Description:

Get members from a sequence start from a specified position, so as to create a new sequence.

Syntax:

A.**to(***a***)** From sequence *A*, get a sequence composed of the first *a* members.

A.**to(***a*,*b***)** From the sequence *A*, get a sequence composed of the members from a^{th} to the b^{th}. If omitting *a*, then return 1 by default; If omitting *b*, then return ***A*.len()** by default. Please note that there is a comma that cannot be omitted.

Remark:

From the sequence *A*, get a sequence composed of the members from a^{th} to the b^{th}. If omitting *a*, then return 1 by default; If omitting *b*, then return ***A*.len()** by default.

Parameters:

A A sequence

a the starting integer

b the ending integer

Return value:

Sequence

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	91	85	98
2		3	==[B2:F2].to(2,5)	==[B2:F2].to(2)				

The value of **A3** is: **[80,91,85,98]**

The value of **B3** is: **[87,80]**

Related concepts:

to()

top()

A.top()

Description:

Get the top n smallest records of sequence members

Syntax:

A. **top**(*n*,*x*,…)

Remark:

x is the expression, based on which each member of a sequence is computed. The records corresponding to the *n* smallest values will be returned. *n* must not be omitted. The omission of *x* is equivalent to ~.

Parameter:

A	A sequence
x	Sort expression
n	Integer

Return value:

Corresponding records whose computational results of *x* are the top n smallest values

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	81
	1	4	David	75	92	89	96	87
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8		==[B2].top(3,~)				

B8 result: [65,75,75]

Related concepts:

A.pos()

A.sort()

A.pos(x)

A.psort()

A.ptop()

topx()

A.topx()

Description:

Get the top *n* smallest values of the sequence.

Syntax:

A.**topx**(*n*,*x*)

Remark:

x is the expression based on which each member of a sequence is computed. Then, the resulting *n* smallest values will be returned. *n* must not be omitted. The omission of *x* is equivalent to ~.

Parameter:

A A sequence

x Sort expression

n Integer

Return value:

The top n smallest values resulting from the formula

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	81
	1	4	David	75	92	89	96	87
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8		==[B2].topx(3, ~+10)				

B8 result: [75,85,85]

Related concepts:

A.pos()

A.sort()

A.pos(x)

A.psort()

A.ptop()

A.top()

trim()

Description:

Remove the space on both ends of a string

Syntax:

trim(*s*)

Remark:

Remove the space on both ends of a string *s*.

Parameters:

s Source string from which you want to remove the space

Options:

@l Remove the spaces on the left of the string *s*, and the option is letter **l**

@r Remove the spaces on the right of string *s*

The default actions are to remove the space on both ends

Return value:

String

Example:

- **trim(" abc ")** **"abc"**
- **trim(" a bc ")** **"a bc"**
- **trim@l(" abc def ")** **"abc def "**
- **trim@l("def abc ")** **"def abc "**
- **trim@r(" abc def ")** **" abc def"**
- **trim@r("def abc ")** **"def abc"**

true

Description:

Logical constants. True value

Syntax:

true

Remark:

It can be used directly in the constant cell or expression.

Example:

0	1	A	B
1-		98	60
	1	==A1>B1	

The value of **A2** is **true**

Related concepts:

null

false

union()

A.union()

Description:

Merge all the members in a sequence whose members may also be sequence.

Syntax:

A.**union()**

Remark:

Generate a new sequence by merging all the members in sequence *A* whose members may also be

sequence.

Duplicate members in latter sequence will be ignored and the duplicate members in the same sequence are not regarded as duplicate members

Parameters:

A A sequence whose members are sequences

Return value:

The new sequence by merging all the members in sequence A

Example:

0	1		A	B
1-		1	Student	English
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	Math
	1	9	Vincent	100
	1	10	Aaron	95
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==[{A2}(to(3)),{A9}(to(3))].union()	

The student whose math or/and English score the top 3

The value of **A15** is: **["Aaron","Charles","David","Vincent"]**, **"Aaron"** and **"Charles"** will not appear repeatedly.

Related concepts:

A.conj()

A.diff()

A.isect()

A.union(x)

Description:

Compute *x* with each member of the sequence whose members are sequences, and then perform union operation on members of the new sequence

Synatax:

A.**union(x)**

Remark:

Compute *x* on sequence *A*, whose members are sequences, by loop, and then perform union operation on members of the resulting sequence

Parameters:

A A sequence whose members are sequences

x an expression, "**~**" in which is used to reference the current member.

Return value:

A new sequence created through the union of sequence *A*

Example:

0	1		A	B
1-		1	Student	Math
	1	2	Aaron	98
	1	3	Charles	95
	1	4	David	87
	1	5	Mary	83
	1	6	Vincent	75
	1	7	Lucy	65
2-		8	Student	PE
	1	9	Vincent	100
	1	10	Aaron	98
	1	11	Charles	92
	1	12	Lucy	88
	1	13	David	80
	1	14	Mary	71
3		15	==[{B2},{B9}].union(~-10)	

A15 results: [88,85,77,73,65,55,90,82,78,70,61]

Related concepts:

A.union()

upper()

Description:

Convert all characters to upper case

Syntax:

upper(*s*)

Remark:

Convert all characters to upper case

Parameters:

s Source string to be converted to upper case

Return value:

String

Example:

- **upper("ABCdef")** **"ABCDEF"**
- **upper("abcDEF")** **"ABCDEF"**

Related concepts:

lower()

Value assignment and computation

Description:

Assign the result of an expression to a variable and return the result of the expression.

Syntax:

a=*x*

Remark:

Assign the result of expression *x* to variable *a* and return the result of the expression *x*.

Parameters:

a The variable name

x The valid expression

Return value:

The result of an expression

Example:

0	1		A	
1-		1	=time=now()	
	1	2	=arg1=5*3	
2		3	=time	The current time
3		4	=arg1	15

variance()

A.variance()

Description:

Compute the variance value of all the non-null members in a sequence.

Syntax:

A.**variance()**

Remark:

Compute the variance of all the non-null members in the sequence *A*.

Parameters:

A an *n* sequence

Return value:

The variance of all the non-null members in the sequence *A*

Example:

0	1		A	B	C	D	E	F	G

1-		1	Student	PE	Math	English	History	Geography	variance
	1	2	Aaron	87	80	98	80	98	==[B2:F2].variance()
	1	3	Charles	90	99	80	76	91	==[B3:F3].variance()
	1	4	David	75	92	89	96	84	==[B4:F4].variance()
	1	5	Mary	93	78	81	92	76	==[B5:F5].variance()
	1	6	Vincent	75	90	88	92	97	==[B6:F6].variance()
	1	6	Lucy	65	71	89	69	92	==[B7:F7].variance()

G2-G7 results: 65.44,67.76,52.56,50.80,53.84,122.56

Related concepts:

A.sum()

A.avg()

A.min()

A.max()

A.count()

A.variance(x)

Description:

Compute *x* with each member of the sequence and then compute the variance value of the members of the new sequence

Syntax:

A.**variance**(*x*) Equivalent to *A*.(*x*).**variance()**

Remark:

Compute *x* on sequence *A* by loop and then compute the variance value of members in the resulting sequence

Parameters:

A A sequence

x an expression, "**~**" in which is used to reference the current member.

Return value:

A numerical value

Example:

0	1		A	B	C	D	E	F
1-		1	Student	PE	Math	English	History	Geography
	1	2	Aaron	87	80	98	80	98
	1	3	Charles	90	99	80	76	91
	1	4	David	75	92	89	96	84
	1	5	Mary	93	78	81	92	76
	1	6	Vincent	75	90	88	92	97
	1	7	Lucy	65	71	89	69	92
2		8		==[B2].variance(~+10)				

B8 results: 98.13888888888808

Related concepts:

A.variance()

words()

s.words()

Description:

Select the English words out of a string

Syntax:

s.**words()**

Remark:

Select the English words out of a string as a sequence of strings and retrun it; other characters will be ingnored.

Options:

@d Select the numbers out of the string *s*

@a Select both the English words and the numbers out of the string *s*

Parameters:

s A string

Return value:

A sequence of strings

Example:

0	1		A	B
1-		1	4,23,a,test?my_file 57	
	1	2	=A1.words()	
2		3	=A1.words@d()	=A1.words@a()

A2 results: ["a","test","my","file"]

A3 results: [4,23,57]

B3 results: [4,23,"a","test","my","file",57]

workday()

Description:

Compute a date time of n workdays from the specified date

Syntax:

workday (*t*,*k*,*h*)

Remark:

Compute a date of k workdays to the date t. The h is (not) a holiday sequence, that is, the member of h is either weekend or holidays. If it is weekend, then swap this day with a workday.

Parameters:

t Date

k Integer

h Time sequence

Return value:

Date time

Example:

- **workday(date("2011-11-07"),25,[date("2011-12-03"),date("2011-12-31")])** **2011-12-09**
- **workday(date("2011-11-07"),25,[date("2011-11-30"),date("2011-12-31")])** **2011-12-13**

workdays()

Description:

Return a sequence of workdays between two dates inclusive

Syntax:

workdays(*b*,*e*,*h***)**

Remark:

Get a sequence of workdays between date *b* and date *e* inclusive. Members of *h* are either weekend or holidays. Depending on what they are, *h* is (or isn't) a sequence of holidays. If there is an off-duty shift in a weekend, take it as the weekdays.

Parameters:

b Date

e Date

h A sequence composed of data of date type

Return value:

A sequence

Example:

- **=workdays(date("2015-04-02"),date("2015-04-08"),[date("2015-04-04"),date("2015-04-05"),date("2015-04-06")])** **[2015-04-02,2015-04-03,2015-04-04,2015-04-05,2015-04-07,2015-04-08]**

xor()

Description:

Perform XOR operation on integers

Syntax:

xor($x_{i,...}$**)**
xor(*A***)**

Remark:

Perform XOR operation on integers

Parameter:

A Sequence
x_i The numerical expresion based on which you perform the XOR operation

Return value:

An integer

Example:

xor(6,11) **13**

year()

Description:

Get the year from a date

Syntax:

year(*dateExp***)**

Remark:

Get the year from the date *dateExp*

Parameters:

dateExp Expression whose result is a date or date time

Return value:

Integer

Example:

- **year(datetime("19800227","yyyyMMdd"))** **1980**
- **year("1972-11-08 10:20:30")** **1972**
- **year(datetime("2006-01-15 13:20:45"))** **2006**

Related concepts:

month()
day()
hour()
minute()
second()
millisecond()