Guide to Calculations and Filters

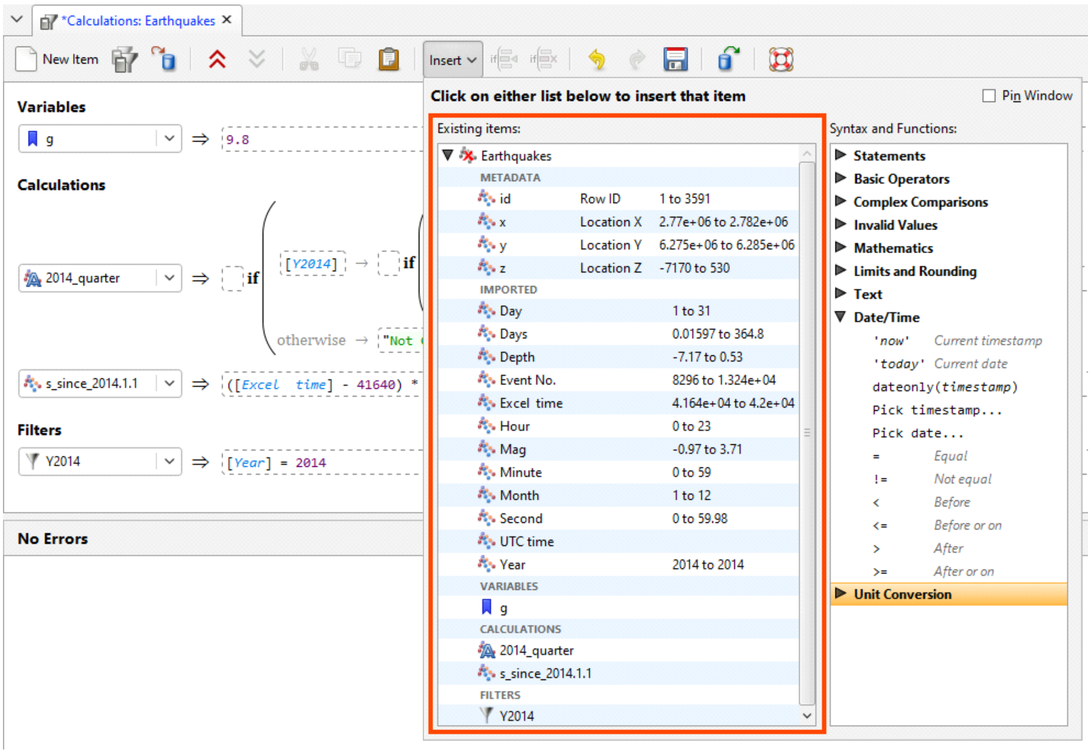
for Leapfrog Geo Version 5.1

Catalogue of Metadata, Syntax and Functions

This catalogue details each of the items in the pinnable **Insert** list for **Calculations and Filters**. Each item includes an intentionally trivial example to illustrate the use of the item, along with an explanation of the effect of the expression.

Existing Items

This section covers the items listed in the left-hand side of the pinnable **Insert** list:



g ⇒ 9.8

2014_quarter ⇒ if [Y2014] → if … otherwise → "Not

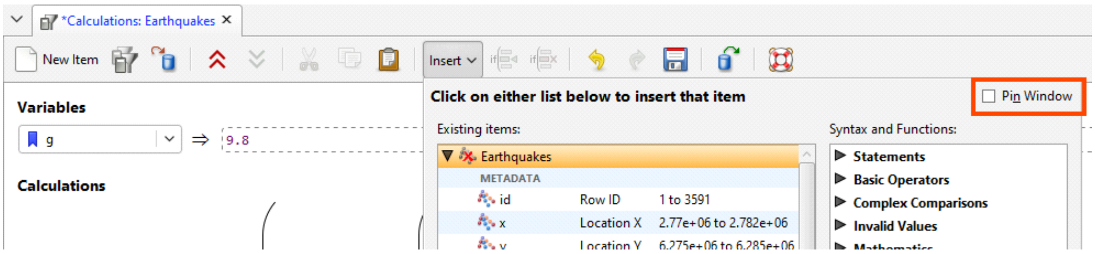
s_since_2014.1.1 ⇒ ([Excel time] - 41640) *

Y2014 ⇒ [Year] = 2014

Existing items:

Earthquakes		
METADATA		
id	Row ID	1 to 3591
x	Location X	2.77e+06 to 2.782e+06
y	Location Y	6.275e+06 to 6.285e+06
z	Location Z	-7170 to 530
IMPORTED		
Day		1 to 31
Days		0.01597 to 364.8
Depth		-7.17 to 0.53
Event No.		8296 to 1.324e+04
Excel time		4.164e+04 to 4.2e+04
Hour		0 to 23
Mag		-0.97 to 3.71
Minute		0 to 59
Month		1 to 12
Second		0 to 59.98
UTC time		
Year		2014 to 2014
VARIABLES		
g		
CALCULATIONS		
2014_quarter		
s_since_2014.1.1		
FILTERS		
Y2014		

To pin these lists to the **Calculations** tab, enable the **Pin Window** option:

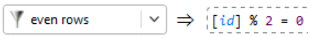


Metadata

id

This metadata item is available for imported points objects. It is the row ID from the points table.

Example



even rows ⇒ [id] % 2 = 0

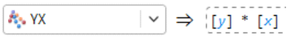
Explanation

The filter **even rows** will select for points with a row **[id]** that has no remainder when divided by 2.

x, y and z

These three metadata items are the variables for locating each point in a points object in X, Y and Z coordinates. Select an item to add it to the expression at the insertion point. Note that whenever one of these metadata items is added to an expression, it is wrapped in square brackets. This is not available for block objects; use xc, xy and xz instead.

Example



YX ⇒ [y] * [x]

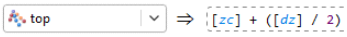
Explanation

The numeric calculation **YX** will be assigned the value of the location of **[y]** multiplied by the location of [x].

xc, yc and zc

These three metadata items are the variables for locating the centroid of each block, in X, Y and Z coordinates. Select an item to add it to the expression at the insertion point. Note that whenever one of these metadata items is added to an expression, it is wrapped in square brackets. This is not available for points objects; use x, y and z instead.

Example



top ⇒ [zc] + ([dz] / 2)

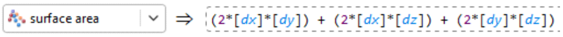
Explanation

The numeric calculation **top** will be assigned the value of the location of **[zc]** the altitude of each block centroid from the zero reference, plus half the height of the block.

dx, dy and dz

These three metadata items are the variables for the block dimensions in X, Y and Z coordinates. Select an item to add it to the expression at the insertion point. Note that whenever one of these metadata items is added to an expression, it is wrapped in square brackets. This is not available for points objects.

Example



surface area ⇒ (2*[dx]*[dy]) + (2*[dx]*[dz]) + (2*[dy]*[dz])

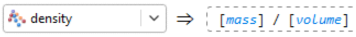
Explanation

The numeric calculation **surface area** will be calculated by figuring the area of each face of the block by multiplying the X and Y dimensions, X and Z dimensions and Y and Z dimensions and adding them together.

volume

This metadata item provides the volume for each block. Note that when this metadata item is added to an expression, it is wrapped in square brackets. This is not available for points objects.

Example



density ⇒ [mass] / [volume]

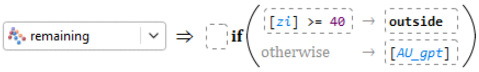
Explanation

The numeric calculation **density** will be assigned the value of the variable **[mass]** divided by the metadata item **[volume]**.

xi, yi and zi

These three metadata items are the variables for locating each block by X, Y and Z index. Select an item to add it to the expression at the insertion point. Note that whenever one of these metadata items is added to an expression, it is wrapped in square brackets. This is not available for points objects.

Example



remaining ⇒ if ([zi] >= 40 → outside
otherwise → [AU_gpt])

Explanation

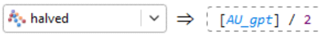
The numeric calculation **remaining** will be assigned the value of the block's **AU_gpt** evaluation, unless **[zi]** the Z index of the block is greater than or equal to 40, in which case the block status will be set to the invalid value **outside**.

Evaluations

Each of these items are automatically added to the list whenever an evaluation is added to the object. When added to an expression, the item represents a placeholder in the calculation for an estimated value, as the expression is evaluated for each of the locations in turn.

Note you can expand each evaluation in the list to see the attributes for the estimation evaluation that may also be selected instead of or in addition to the estimated value.

Example



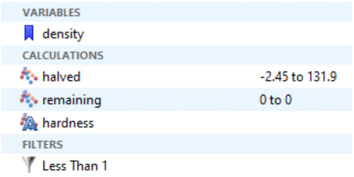
```
halved ⇒ [AU_gpt] / 2
```

Explanation

The numeric calculation **halved** is defined by the evaluation **[AU_gpt]** divided by 2. Each location in the object will have its own value for **AU_gpt**, and this calculation uses those values to create a new value named **halved**for each location, using the formula above.

Variables, Calculations and Filters

Each time you create a new variable, numeric calculation, category calculation, or filter in **Calculations**, it will also be added to the **Existing items** list. You can select them from this list and they will be inserted into your new expression at the insertion point. Note that whenever one of these named items is added to an expression, it is wrapped in square brackets.



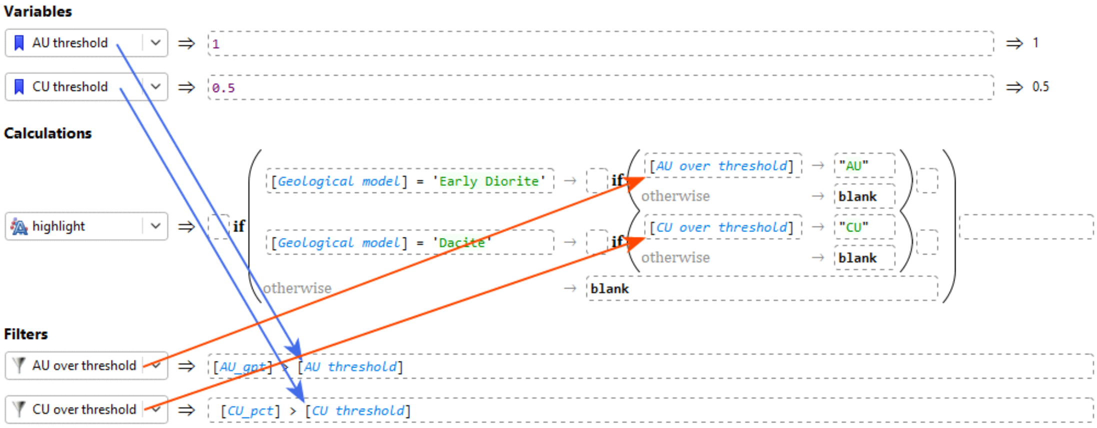
VARIABLES
- density

CALCULATIONS

halved	-2.45 to 131.9
remaining	0 to 0
hardness	

FILTERS
- Less Than 1

Example



Variables

```
AU threshold ⇒ 1 ⇒ 1
CU threshold ⇒ 0.5 ⇒ 0.5
```

Calculations

```
highlight ⇒ if ( [Geological model] = 'Early Diorite' → if ( [AU over threshold] → "AU"
                                                              otherwise → blank )
                 [Geological model] = 'Dacite' → if ( [CU over threshold] → "CU"
                                                       otherwise → blank )
                 otherwise → blank )
```

Filters

```
AU over threshold ⇒ [AU_gpt] > [AU threshold]
CU over threshold ⇒ [CU_pct] > [CU threshold]
```

Explanation

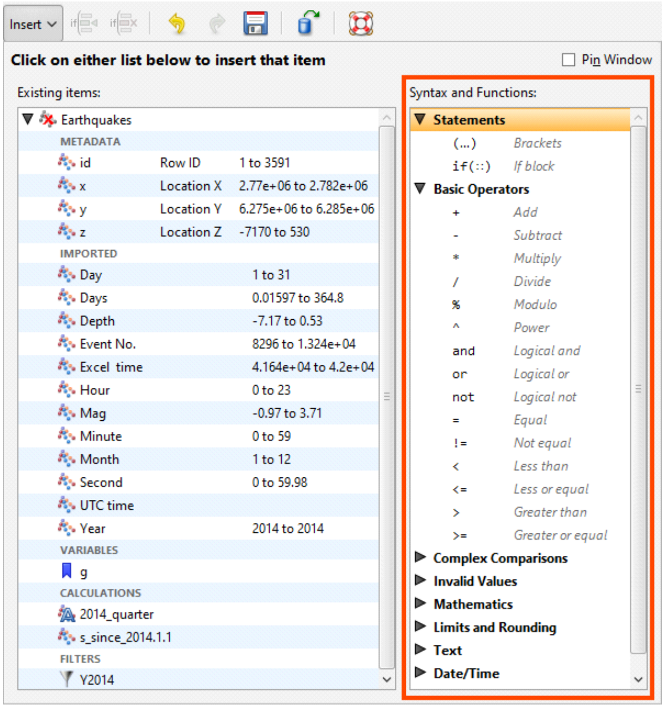
This example is only attempting to illustrate how variables, calculations and filters that have already been defined can be referenced by name in new calculations; the calculation **highlight** is not intended to be useful.

It is good practice to break your calculations down into parts, giving each part a relevant and readily identifiable, unambiguous and easily understood name. This will make your calculations more readable and clear. Using **Variables** to define a constant with a name makes it easy to understand the utility of that particular constant when you use it in a calculation. You may also be able to re-use certain parts such as filters or constant variables, so you do not need to define the same thing repeatedly.

Be careful not to inadvertently name something incorrectly, such as naming a volume as "area", as this could give rise to difficult-to-locate errors in your calculation.

Syntax and Functions

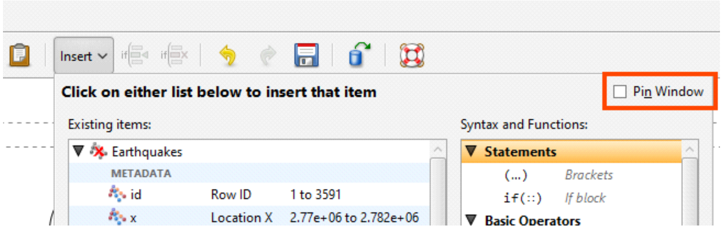
This section covers the items listed in the right-hand side of the pinnable **Insert** list.



Earthquakes		
METADATA		
id	Row ID	1 to 3591
x	Location X	2.77e+06 to 2.782e+06
y	Location Y	6.275e+06 to 6.285e+06
z	Location Z	-7170 to 530
IMPORTED		
Day		1 to 31
Days		0.01597 to 364.8
Depth		-7.17 to 0.53
Event No.		8296 to 1.324e+04
Excel time		4.164e+04 to 4.2e+04
Hour		0 to 23
Mag		-0.97 to 3.71
Minute		0 to 59
Month		1 to 12
Second		0 to 59.98
UTC time		
Year		2014 to 2014
VARIABLES		
g		
CALCULATIONS		
2014_quarter		
s_since_2014.1.1		
FILTERS		
Y2014		

Statements	
(...)	*Brackets*
if(::)	*If block*
Basic Operators	
+	*Add*
-	*Subtract*
*	*Multiply*
/	*Divide*
%	*Modulo*
^	*Power*
and	*Logical and*
or	*Logical or*
not	*Logical not*
=	*Equal*
!=	*Not equal*
<	*Less than*
<=	*Less or equal*
>	*Greater than*
>=	*Greater or equal*
Complex Comparisons	
Invalid Values	
Mathematics	
Limits and Rounding	
Text	
Date/Time	

To pin these lists to the **Calculations** tab, enable the **Pin Window** option:

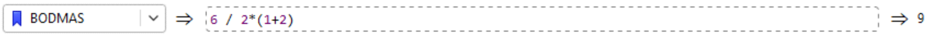


Statements

(...) *Brackets*

Brackets are used to enclose an expression so the operations on the values within the brackets take precedence over operations outside the brackets.

Example



```
BODMAS ⇒ 6 / 2*(1+2) ⇒ 9
```

Explanation

1+2 will be calculated prior to calculating the result of the expression, following the standard order of mathematical operations.

if(::) *If block*

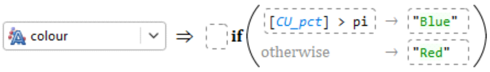
The **If block** is used for conditional logic. This allows multiple pathways to results depending on selected conditions, or categorisation based on values.

An **If block** will be evaluated by each test → result, row by row, separately and in order downwards by row. Each test has an output that can be **'true'**, **'false'**, or an invalid value **'error'**, **'blank'**, **'without_value'** or **'outside'**. The result output is produced by the execution of the result expression. The **If block** output follows these rules:

- If a test output is **'error'**, the **If block** output is **'error'** and no further processing of subsequent rows is done.
- If a test output is **'false'**, the result expression is not executed, and the next row is considered.
- If a test is **'true'**, the result expression is executed and the result output is used for the **If block** output and no further processing of subsequent rows is done.
- If all the tests are **'false'**, the **'otherwise'** result expression is executed and its output is used.
- If a test output is an invalid value (**without_value**, **blank**, **outside**) the result expression is not executed and the test output's invalid value is remembered. The subsequent rows are then run.
- If a subsequent test output is **'true'** after an earlier one produced an invalid value, the previous test output is discarded and the new row's result expression output is used as the output for the **If block**.
- If all test outputs are invalid status values, the highest priority status of all the remembered invalid statuses is used as the output result. The priority of non-error invalid status values is: **outside** > **without_value** > **blank**.

Additionally, it is possible to choose to produce invalid values as the output of result expressions.

Example

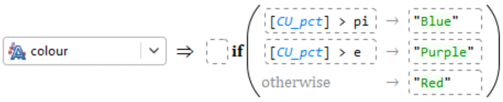


```
colour ⇒ if( [CU_pct] > pi → "Blue"
             otherwise     → "Red" )
```

Explanation

Cu_pct is the name of an evaluation applied to all the points in a points object. As the **if(::) If block** calculation is run for each point, the evaluation for each point replaces this variable name in the expression. If the value is greater than the value of the constant **pi**, the result for that point will be the text string **"Blue"**. Otherwise, if the value is less than or equal to **pi**, the result will be the string **"Red"**.

Additional rows may be added. Each row follows on from the left-over results of the line before, simplifying the logical expression that may be used.

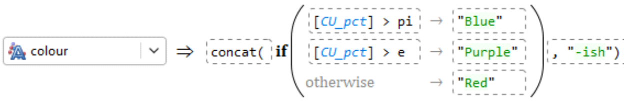


```
colour ⇒ if( [CU_pct] > pi → "Blue"
             [CU_pct] > e  → "Purple"
             otherwise     → "Red" )
```

Explanation

Note the addition of the line **[Cu_pct] > e → "Purple"**. This line can be interpreted to mean: if the value estimated for a point is less than or equal to **pi**, but greater than **e** the result shall be **"Purple"**. The part about it being less than or equal to pi is implied because the line follows the previous line **[Cu_pct] > pi → "Blue"**.

Note that expression elements before and after the if expression can be entered. This allows the **if(::) If block** to form part of a more complex or extensive expression.



```
colour ⇒ concat( if ⎛ [CU_pct] > pi → "Blue"     ⎞ , "-ish")
                    ⎜ [CU_pct] > e  → "Purple"   ⎟
                    ⎝ otherwise     → "Red"      ⎠
```

Explanation

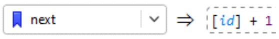
The earlier conditional classification has now been embedded within a concatenation function, forming the first part of a two-part string concatenation. The concatenation function is adding the text string **"-ish"** to whatever is produced by the **if(::) If block**. Thus, if the value of the block being evaluated is 1.2, the result of the **colour** categorisation calculation will be **"Red-ish"**.

Basic Operators

+ Add

An arithmetic addition operation.

Example



```
next ⇒ [id] + 1
```

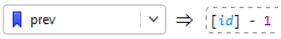
Explanation

next is assigned the value of the current imported points row **[id]** plus 1.

- Subtract

An arithmetic subtraction operation.

Example



```
prev ⇒ [id] - 1
```

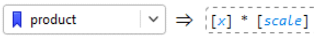
Explanation

prev is assigned the value of the current imported points row **[id]** minus 1.

* *Multiply*

An arithmetic multiplication operation. Note that implied multiplication, putting factors adjacent to one another, is not supported. The ***** operator must be explicitly used.

Example



product ⇒ [x] * [scale]

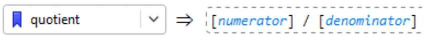
Explanation

product is assigned the value of **[x]** times **[scale]**.

/ *Divide*

An arithmetic division operation.

Example



quotient ⇒ [numerator] / [denominator]

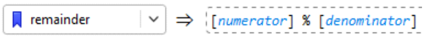
Explanation

quotient is assigned the value of **[numerator]** divided by **[denominator]**.

% *Modulo*

An arithmetic modulo operation. This is an integer division operation that returns the remainder instead of the integer quotient.

Example



remainder ⇒ [numerator] % [denominator]

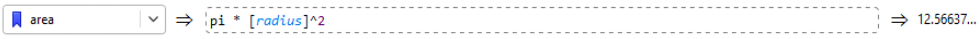
Explanation

remainder is assigned the value of **[numerator]** modulo **[denominator]**, or in other words, **[numerator]** is divided by **[denominator]** to produce an integer quotient , the number of times **[denominator]** goes into **[numerator]**, and a remainder, which is the number returned by this modulo function.

^ *Power*

A mathematical exponentiation operation where a base is raised to the power of the exponent.

Example



area ⇒ pi * [radius]^2 ⇒ 12.56637...

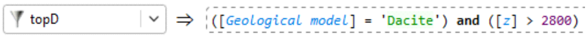
Explanation

area is assigned the value of **pi** * **[radius]**^{2} (because **^2** is interpreted as ‘to the power of the exponent 2’ or 'squared'). Because **[radius]** happened to be defined as equalling 2, the result of the expression **pi** * 2^2 is 12.56637..., as can be seen from the result at the end of the expression.

and *Logical and*

A logical **and** operation.

Example



```
topD ⇒ ([Geological model] = 'Dacite') and ([z] > 2800)
```

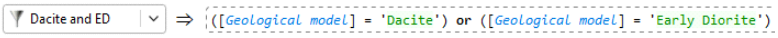
Explanation

filter will be true if the point is classified in the **'Dacite'** part of the geological model AND the **[z]** coordinate for the point is above 2800; it will be false if either of these conditions are not true.

or *Logical or*

A logical **or** operation.

Example



```
Dacite and ED ⇒ ([Geological model] = 'Dacite') or ([Geological model] = 'Early Diorite')
```

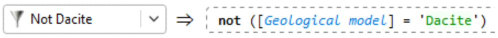
Explanation

Dacite and ED will be true if the point is classified in the **'Dacite'** part of the geological model OR the the point is classified in the **'Early Diorite'** part of the geological model, but it will be false if neither of these conditions is true.

not *Logical not*

A logical **not** operation.

Example



```
Not Dacite ⇒ not ([Geological model] = 'Dacite')
```

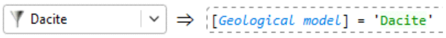
Explanation

Not Dacite will be true if the point is classified in the geological model as anything other than **'Dacite'**. The logical operator **not** inverts the logical expression that follows the operator.

= *Equal*

A logical equality operator.

Example



Dacite ⇒ `[Geological model] = 'Dacite'`

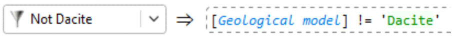
Explanation

Dacite will be true if the point is classified in the **'Dacite'** part of the geological model, and will be false for all other values.

!= *Not equal*

A logical not-equal operator.

Example



Not Dacite ⇒ `[Geological model] != 'Dacite'`

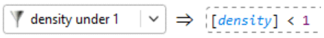
Explanation

Not Dacite will be true if the point is classified in the geological model as anything other than **'Dacite'**, and will be false when it is **'Dacite'**.

< *Less than*

A logical less-than operator.

Example



density under 1 ⇒ `[density] < 1`

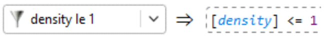
Explanation

density under 1 will be true when the variable **[density]** is less than 1, and false otherwise.

< *Less or equal*

A logical less-than-or-equals operator.

Example



density le 1 ⇒ `[density] <= 1`

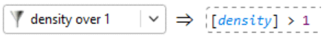
Explanation

density le 1 will be true when the variable **[density]** is less than or equal to 1, and false otherwise.

< *Greater than*

A logical greater-than operator.

Example



density over 1 ⇒ [density] > 1

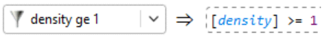
Explanation

density over 1 will be true when the variable **[density]** is more than 1, and false otherwise.

< *Greater or equal*

A logical greater-than-or-equals operator.

Example



density ge 1 ⇒ [density] >= 1

Explanation

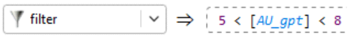
density ge 1 will be true when the variable **[density]** is more than or equal to 1, and false otherwise.

Complex Comparisons

Lower < *n* < upper

A pair of comparisons, with a logical result to indicate if the tested value **n** is between the **lower** value provided and the **upper** value provided.

Example



filter ⇒ 5 < [AU_gpt] < 8

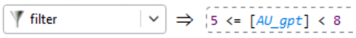
Explanation

filter will be true when the point evaluation **[AU_gpt]** is between the values of 5 and 8 (but not equalling 5 or 8); it will be false otherwise.

Lower <= *n* < upper

A pair of comparisons, with a logical result to indicate if the tested value **n** is between the **lower** value provided and the **upper** value provided, or equal to the **lower** value.

Example



filter ⇒ 5 <= [AU_gpt] < 8

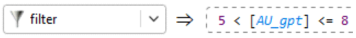
Explanation

filter will be true when the point evaluation **[AU_gpt]** is between the values of 5 and 8 (but not equalling 8); it will be false otherwise.

Lower < *n* <= upper

A pair of comparisons, with a logical result to indicate if the tested value **n** is between the **lower** value provided and the **upper** value provided, or equal to the **upper** value.

Example



filter ⇒ 5 < [AU_gpt] <= 8

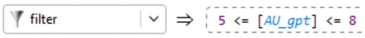
Explanation

filter will be true when the point evaluation **[AU_gpt]** is between the values of 5 and 8 (but not equalling 5); it will be false otherwise.

Lower <= *n* <= upper

A pair of comparisons, with a logical result to indicate if the tested value **n** is between (or equal to either) the **lower** value provided and the **upper** value provided.

Example



filter ⇒ 5 <= [AU_gpt] <= 8

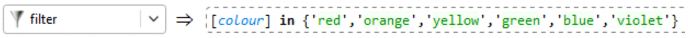
Explanation

filter will be true when the point evaluation **[AU_gpt]** is greater than or equal to 5 and less than or equal to 8; it will be false otherwise.

x in {a,b,...}

A logical inclusion expression. This will return true if **x** matches any element of the set of listed items.

Example



```
filter ⇒ [colour] in {'red','orange','yellow','green','blue','violet'}
```

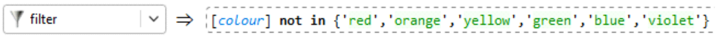
Explanation

If **[colour]** is one of the items in the list within the curly brackets, **filter** will be true. If **[colour]** is anything else, **filter** will be false.

x not in {a,b,...}

A logical exclusion expression. This will return true if **x** fails to match any element of the set of listed items.

Example



```
filter ⇒ [colour] not in {'red','orange','yellow','green','blue','violet'}
```

Explanation

If **[colour]** is not one of the items in the list within the curly brackets, **filter** will be true. If **[colour]** matches any item in the list, **filter** will be false.

Invalid Values

Invalid values are different types of results for “numeric” categorisation calculations that need special non-numeric results for certain category results. These have special meanings of their own without having to resort to interpreting negative numbers and zero as having special meaning.

blank means having no value, the value in the imported file is blank or has non-numeric data

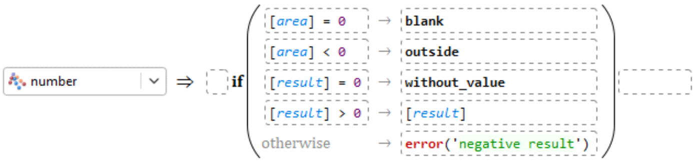
without_value is often used to mean the estimator cannot produce a value (specific to blocks)

outside is used to indicate the block is outside the boundary of the domain (specific to blocks)

error generates an error, and provides an ‘error’ status value for the affected block or point

error(‘message’) is similar to **error** but includes a custom message.

Multiple case example



```
number ⇒ if ( [area] = 0      → blank
              [area] < 0      → outside
              [result] = 0    → without_value
              [result] > 0    → [result]
              otherwise       → error('negative result') )
```

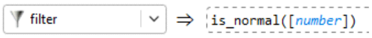
Explanation

If the variable **[area]** is equal to 0, the point will be marked with the special value **blank**. If the variable **[area]** is less than 0, the point will be marked with the special value **outside**. If **[area]** is greater than 0 and the variable **[result]** is equal to 0, the point will be marked with the special value **without_value**. If **[area]** is greater than 0 and **[result]** is greater than 0, the point will be will be assigned the value of the variable **[result]**. If **[area]** is greater than 0 and **[result]** is less than 0, the point will be given the special value **error** and status with the message **'negative result'**.

is_normal(*a*)

A function that tests **a** to see if it is a normal value or an invalid value. If **a** is normal, it returns true. If **a** is invalid, it returns false.

Example



filter ⇒ is_normal([number])

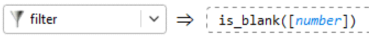
Explanation

If **[number]** has a normal value, **filter** will be true for that block. If it produces an invalid value, **filter** will be false.

is_blank(*a*)

A function that tests **a** to see if it is a **blank** invalid value. If **a** is **blank**, it returns true. If **a** is normal or another invalid value, it returns false.

Example



filter ⇒ is_blank([number])

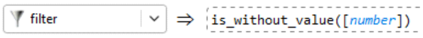
Explanation

If **[number]** has a **blank** status, **filter** will be true for that block. If it produces any other invalid value or a normal value, **filter** will be false.

is_without_value(*a*)

A function that tests **a** to see if it is a **without_value** invalid value. If **a** is **without_value**, it returns true. If **a** is normal or another invalid value, it returns false.

Example



filter ⇒ is_without_value([number])

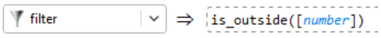
Explanation

If **[number]** has a **without_value** status, **filter** will be true for that block. If it produces any other invalid value or a normal value, **filter** will be false.

is_outside(*a*)

A function that tests **a** to see if it is an **outside** invalid value. If **a** is **outside**, it returns true. If **a** is normal or another invalid value, it returns false.

Example



```
filter ⇒ is_outside([number])
```

Explanation

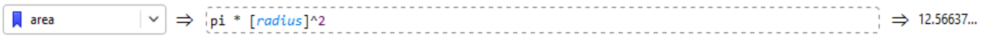
If **[number]** has an **outside** status, **filter** will be true for that block. If it produces any other invalid value or a normal value, **filter** will be false.

Mathematics

pi *Constant*

The constant **pi** is an existing item you do not need to define yourself. It is defined to 15 decimal places as 3.141592653589793.

Example



```
area ⇒ pi * [radius]^2 ⇒ 12.56637...
```

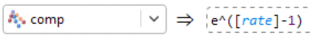
Explanation

area will be calculated as **pi** multiplied by the square of **[radius]**. Note that **pi** is not enclosed in square brackets like user-created variables and constants, as it is an internal constant.

e *Constant*

The constant **e**, the base of the natural logarithm, is an existing item you do not need to define yourself. It is defined to 15 decimal places as 2.718281828459045.

Example



```
comp ⇒ e^([rate]-1)
```

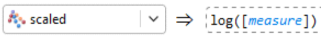
Explanation

comp will be calculated as **e** to the power of (**[rate]** minus 1). Note that **e** is not enclosed in square brackets like user-created variables and constants, as it is an internal constant.

log(*n*) *Base 10*

The common logarithm, the logarithm with base 10, i.e. log$_{10}$(n) or lg(n). This function will calculate the common logarithm of the value provided as **n**.

Example



scaled ⇒ `log([measure])`

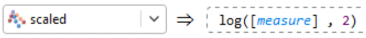
Explanation

scaled will be calculated as log$_{10}$ of **[measure]**.

log(*n, base*)

The logarithm of a number **n** to the base **base**.

Example



scaled ⇒ `log([measure] , 2)`

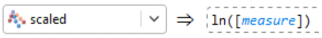
Explanation

scaled will be calculated as **log$_2$** of **[measure]**.

ln(*n*) *Natural log, base e*

The natural logarithm, the logarithm with base e, i.e. log$_e$(n). This function will calculate the natural logarithm of the value provided as **n**.

Example



scaled ⇒ `ln([measure])`

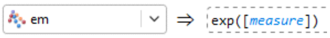
Explanation

scaled will be calculated as log$_e$ of **[measure]**.

exp(*n*) *Natural exponent*

The natural exponent. This function will provide the result of e^n.

Example



em ⇒ `exp([measure])`

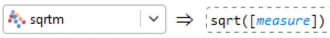
Explanation

em will be calculated as $e^{\textbf{[measure]}}$.

sqrt(*n*) *Square root*

The principle square root of the provided number **n**.

Example



sqrtm ⇒ `sqrt([measure])`

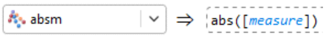
Explanation

sqrtmwill be calculated as the square root of **[measure]**.

abs(*n*) *Absolute value*

The absolute value of a number is it's value with the sign of the number disregarded. The absolute value of -42 is 42. The absolute value of 42 is also 42.

Example



absm ⇒ `abs([measure])`

Explanation

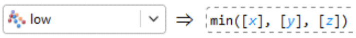
absm will be whatever **[measure]** is, but without its sign; it will always be positive as a result.

Limits and Rounding

min (*n, m, ...*)

Returns the lowest of all the values in the set provided.

Example



low ⇒ `min([x], [y], [z])`

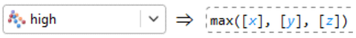
Explanation

low will be the lowest of the three values provided, the metadata items for the X, Y, and Z point coordinates.

max (*n, m, ...*)

Returns the highest of all the values in the set provided.

Example



```
high ⇒ max([x], [y], [z])
```

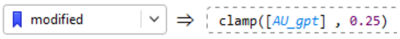
Explanation

high will be the highest of the three values provided, the metadata items for the X, Y, and Z point coordinates.

clamp(*n, lower*)

This clamp function tests the value **n** against the threshold **lower** and if it is less than **lower** the result will be **lower**; otherwise the result will be **n**. The effect is to push all the values below the threshold up to the threshold.

Example



```
modified ⇒ clamp([AU_gpt] , 0.25)
```

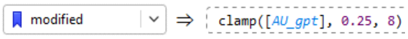
Explanation

The output for **modified** will range from 0.25 up to the maximum value of **[AU_gpt]**.If **[AU_gpt]** is less than 0.25, the output will be 0.25 instead. Otherwise, the output will be **[AU_gpt]**.

clamp(*n, lower, upper*)

This clamp function tests the value **n** against the threshold **lower** and if it is less than **lower** the result will be **lower**; it tests the value **n** against the threshold **upper** and if it is more than **upper** the result will be **upper**; otherwise the result will be **n**. The effect is to squish all the values into a box between the **lower** and **upper** thresholds.

Example



```
modified ⇒ clamp([AU_gpt], 0.25, 8)
```

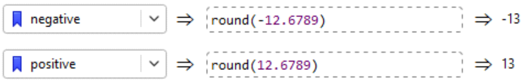
Explanation

The output for **modified** will range from 0.25 up to 8. If **[AU_gpt]** is less than 0.25, the output will be 0.25 instead. If **[AU_gpt]** is more than 8, the output will be 8 instead. Otherwise, the output will be **[AU_gpt]**.

round(*n*)

This function rounds the input value **n** to the nearest whole number.

Example



```
negative  ⇒ round(-12.6789)  ⇒ -13
positive  ⇒ round(12.6789)   ⇒ 13
```

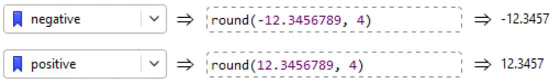
Explanation

The variable **negative** will be given the value -13 as the nearest whole number to **-12.6789** used as the input to the function. The variable **positive** will be given the value 13 as the nearest whole number to **12.6789** used as the input to the function.

round(*n, dp*)

This function rounds the input value **n** to the number of decimal places specified by **dp**, a positive integer.

Example



```
negative  ⇒ round(-12.3456789, 4)  ⇒ -12.3457
positive  ⇒ round(12.3456789, 4)   ⇒ 12.3457
```

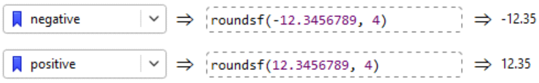
Explanation

The variable **negative** will be given the value -12.3457, the value of **-12.3456789** rounded to 4 decimal places. The variable **positive** will be given the value 12.3457, the value of **12.3456789** rounded to 4 decimal places.

roundsf(*n, sf*)

This function rounds the input value **n** to the number of significant figures specified by **sf**, which must be a positive integer >= 1. Rounding to a given number of significant figures is often preferred in scientific applications over rounding to a given number of decimal places, as outputs can be rounded to the same amount of significance as the inputs.

Example



```
negative ⇒ roundsf(-12.3456789, 4) ⇒ -12.35
positive ⇒ roundsf(12.3456789, 4) ⇒ 12.35
```

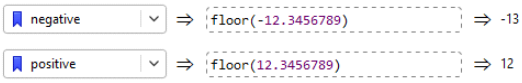
Explanation

The variable **negative** will be given the value -12.35, the value of **-12.3456789** rounded to 4 significant figures. The variable **positive** will be given the value 12.35, the value of **12.3456789** rounded to 4 significant figures.

floor(*n*)

This function removes the fractional part of a real number **n** and returns the integer number below the real number **n**. This remains true when **n** is a negative number.

Example



```
negative ⇒ floor(-12.3456789) ⇒ -13
positive ⇒ floor(12.3456789) ⇒ 12
```

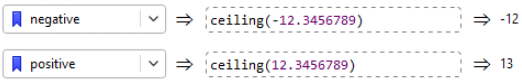
Explanation

The variable **negative** will be given the value -13, the integer below **-12.3456789**. The variable **positive** will be given the value 12, the integer below **12.3456789**.

ceiling(*n*)

This function removes the fractional part of a real number **n** and returns the integer number above the real number **n**. This remains true when **n** is a negative number.

Example



```
negative ⇒ ceiling(-12.3456789) ⇒ -12
positive ⇒ ceiling(12.3456789) ⇒ 13
```

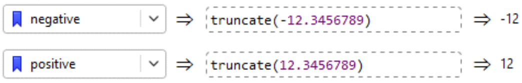
Explanation

The variable **negative** will be given the value -12, the integer above **-12.3456789**. The variable **positive** will be given the value 13, the integer above **12.3456789**.

truncate(*n*)

This function simply removes the fractional part of a real number **n** and returns the integer number without the fractional part. This means that for positive real numbers, the result will be the integer less than the real number **n**, but for negative real numbers, the result will be the integer greater than the real number **n**.

Example



negative ⇒ truncate(-12.3456789) ⇒ -12

positive ⇒ truncate(12.3456789) ⇒ 12

Explanation

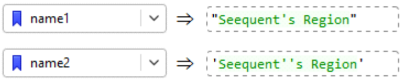
The variable **negative** will be given the value -12, the integer part of **-12.3456789**. The variable **positive** will be given the value 12, the integer part of **12.3456789**.

Text

'abc' Text value

Use this item to add a text sequence to an expression. Selecting the item will add two single quotation marks with the cursor between, ready for the text sequence to be typed. You can of course simply type the quote marks into the expression yourself. Double quotations marks also work identically to the single quotation marks used by this item. If you need to include a quote mark inside your text sequence, you need to "escape" the character so it is not interpreted as the end of the text sequence, by entering two quotation marks for each quotation mark you want inside the text sequence. Alternatively, you can use a different type of quotation mark as the sequence wrappers; for instance to write **Seequent's Region** with an internal apostrophe, you might wrap the sequence with double quotation marks: **"Seequent's Region"**.

Example



name1 ⇒ "Seequent's Region"

name2 ⇒ 'Seequent''s Region'

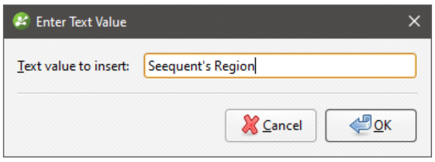
Explanation

The text sequence **Seequent's Region** (note: without the wrapping quotation marks and with only one possessive apostrophe) will be used wherever the variables **name1** or **name2** are used in expressions. Both techniques for including apostrophes in the text sequence have the same result.

Enter text...

This selection opens a dialog box prompting you for text. After you enter it and click OK, the text will be entered at the cursor, wrapped with quotation marks. This is an easy way to resolve any issues about internal quotation marks, as the dialog box will convert the text into the necessary character sequence required to generate your desired text.

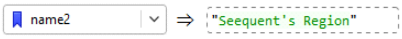
Example



Text value to insert: Seequent's Region

Explanation

This will produce a converted character sequence that produces a valid string and insert it to the expression at the insertion point.

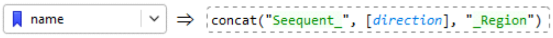


```
name2 ⇒ "Seequent's Region"
```

concat(*t, u, ...*)

This concatenates a series of text sequences together.

Example



```
name ⇒ concat("Seequent_", [direction], "_Region")
```

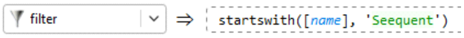
Explanation

Each of the text sequences in the input are run together and combined. If **[direction]** is **'North'** then **name** will be **Seequent_North_Region**. If **[direction]** is **'South'** then **name** will be **Seequent_South_Region**.

startswith(*t, 'prefix'*)

This function returns true if the text sequence **t** starts with **prefix**, and false otherwise. This is case insensitive; 'prefix' will match 'PREFIX'.

Example



```
filter ⇒ startswith([name], 'Seequent')
```

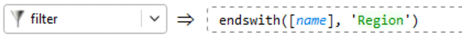
Explanation

filter will be true if **[name]** starts with **Seequent**.

endswith(*t, 'suffix'*)

This function returns true if the text sequence **t** ends with **suffix**, and false otherwise. This is case insensitive; 'suffix' will match 'SUFFIX'.

Example



```
endswith([name], 'Region')
```

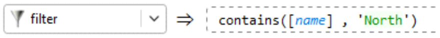
Explanation

filter will be true if **[name]** ends with **Region**.

contains(*t, 'part'*)

This function returns true if the text sequence **t** contains **part** somewhere within, and false otherwise. This is case insensitive; 'part' will match 'PART'.

Example



```
contains([name] , 'North')
```

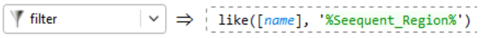
Explanation

filter will be true if **[name]** contains **North** somewhere within the character sequence.

like(*t, 'pattern'*)

This function returns true if the text sequence **t** matches **pattern**, where **[pattern]** is follows **SQL**-style **LIKE** matching rules. This is case insensitive, and pattern must match the whole of **t**, not just a portion of it. Use _ as a wildcard for a single character, and **%** as a wildcard for any number of characters (including no characters).

Example



```
like([name], '%Seequent_Region%')
```

Explanation

filter will be true if **[name]** matches the pattern **%Seequent_Region%**. Examples of [name] that will match include:

- SEEQUENT1Region
- SEEQUENT1REGION
- Seequent2region
- NorthernSeequent1Region
- SEEQUENT3regionExtra
- #seequentXregion#

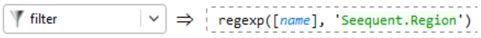
Examples that will not match include:

- Seequent12Region
- SequentReg1ion
- SeequentRegion

regexp(*t*, *'pattern'*)

This function returns true if the text sequence **t** matches **pattern**, where **[pattern]** is follows regular expression matching rules. This is case insensitive.

Example



filter ⇒ `regexp([name], 'Seequent.Region')`

Explanation

filter will be true if **[name]** matches the regexp pattern **Seequent.Region**. Examples of [name] that will match include:

- SEEQUENT1Region
- SEEQUENT1REGION
- Seequent2region
- NorthernSeequent1Region
- SEEQUENT3regionExtra
- #seequentXregion#

Examples that will not match include:

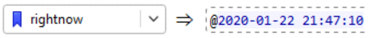
- Seequent12Region
- SeequentReg1ion
- SeequentRegion

Date/Time

'now' Current timestamp

Use this item to add the current date-and-timestamp at the insertion point.

Example



rightnow ⇒ `@2020-01-22 21:47:10`

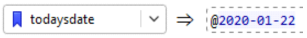
Explanation

While entering an expression, the **'now'** selection has been chosen and a date-and-timestamp has been entered at the insertion point.

'*today*' *Current date*

Use this item to add the current datestamp at the insertion point.

Example



todaysdate ⇒ @2020-01-22

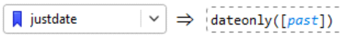
Explanation

While entering an expression, the **'today'** selection has been chosen and a date-and-timestamp has been entered at the insertion point.

dateonly(*timestamp*)

This function takes a date-and-timestamp and cuts off the timestamp to leave just the date.

Example



justdate ⇒ dateonly([*past*])

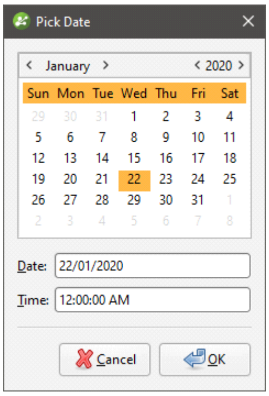
Explanation

While the **[past]** variable has a full date-and-timestamp, the **dateonly** function strips the time off and leaves just the date.

Pick timestamp...

This selection opens a dialog box prompting you for a date and a time that will be entered at the insertion point when you click OK.

Example



Date: 22/01/2020

Time: 12:00:00 AM

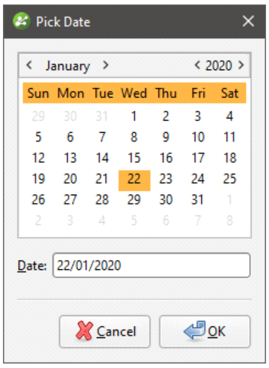
Explanation

Use the year picker, month picker, date selector and **Time** field to specify a date and time. If the above date is entered, it will be represented in the expression as @2020-01-22 00:00:00.

Pick date...

This selection opens a dialog box prompting you for a date that will be entered at the insertion point when you click OK.

Example



Date: 22/01/2020

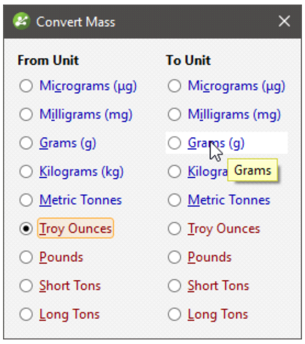
Explanation

Use the year picker, month picker and date selector to specify a date. If the above date is entered, it will be represented in the expression as @2020-01-22.

Unit Conversion

Mass/weight...

This selection opens a dialog box prompting for the **From Unit** and **To Unit** for the conversion.

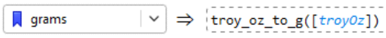


Convert Mass

From Unit	To Unit
○ Micrograms (µg)	○ Micrograms (µg)
○ Milligrams (mg)	○ Milligrams (mg)
○ Grams (g)	○ Grams (g)
○ Kilograms (kg)	○ Kilogra
○ Metric Tonnes	○ Metric Tonnes
● Troy Ounces	○ Troy Ounces
○ Pounds	○ Pounds
○ Short Tons	○ Short Tons
○ Long Tons	○ Long Tons

When you have selected the two units, the dialog box disappears and the conversion function you require will have been inserted into the expression at the cursor, and the cursor repositioned so you can enter the source of the value to be converted.

Example



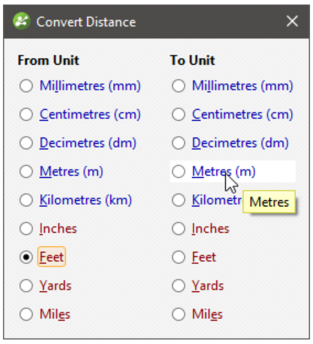
grams ⇒ troy_oz_to_g([troyOz])

Explanation

The **troy_oz_to_g** function has been entered by the dialog box, and the variable **[troyOz]** has been entered as the input. The numeric calculation **grams** will be given the output of the **troy_oz_to_g** function.

Distance...

This selection opens a dialog box prompting for the **From Unit** and **To Unit** for the conversion.

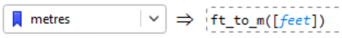


Convert Distance

From Unit	To Unit
○ Millimetres (mm)	○ Millimetres (mm)
○ Centimetres (cm)	○ Centimetres (cm)
○ Decimetres (dm)	○ Decimetres (dm)
○ Metres (m)	○ Metres (m)
○ Kilometres (km)	○ Kilometr
○ Inches	○ Inches
● Feet	○ Feet
○ Yards	○ Yards
○ Miles	○ Miles

When you have selected the two units, the dialog box disappears and the conversion function you require will have been inserted into the expression at the cursor, and the cursor repositioned so you can enter the source of the value to be converted.

Example



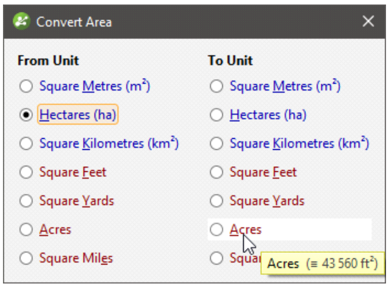
```
metres ⇒ ft_to_m([feet])
```

Explanation

The **ft_to_m** function has been entered by the dialog box, and the variable **[feet]** has been entered as the input. The numeric calculation **metres** will be given the output of the**ft_to_m** function.

Area...

This selection opens a dialog box prompting for the **From Unit** and **To Unit** for the conversion.



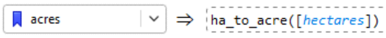
Convert Area

From Unit	To Unit
○ Square Metres (m^2)	○ Square Metres (m^2)
● Hectares (ha)	○ Hectares (ha)
○ Square Kilometres (km^2)	○ Square Kilometres (km^2)
○ Square Feet	○ Square Feet
○ Square Yards	○ Square Yards
○ Acres	○ Acres
○ Square Miles	○ Squa

Acres (≡ 43 560 ft^2)

When you have selected the two units, the dialog box disappears and the conversion function you require will have been inserted into the expression at the cursor, and the cursor repositioned so you can enter the source of the value to be converted.

Example



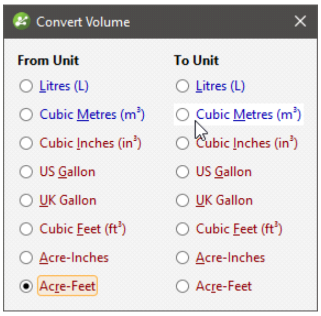
```
acres ⇒ ha_to_acre([hectares])
```

Explanation

The **ha_to_acre** function has been entered by the dialog box, and the variable **[hectares]** has been entered as the input. The numeric calculation **acres** will be given the output of the**ha_to_acre** function.

Volume...

This selection opens a dialog box prompting for the **From Unit** and **To Unit** for the conversion.

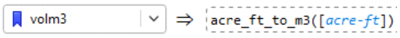


Convert Volume

From Unit	To Unit
○ Litres (L)	○ Litres (L)
○ Cubic Metres (m^3)	○ Cubic Metres (m^3)
○ Cubic Inches (in^3)	○ Cubic Inches (in^3)
○ US Gallon	○ US Gallon
○ UK Gallon	○ UK Gallon
○ Cubic Feet (ft^3)	○ Cubic Feet (ft^3)
○ Acre-Inches	○ Acre-Inches
● Acre-Feet	○ Acre-Feet

When you have selected the two units, the dialog box disappears and the conversion function you require will have been inserted into the expression at the cursor, and the cursor repositioned so you can enter the source of the value to be converted.

Example



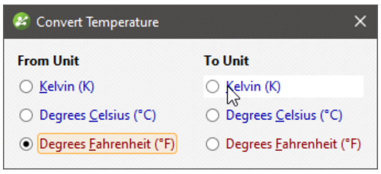
```
volm3 ⇒ acre_ft_to_m3([acre-ft])
```

Explanation

The **acre_ft_to_m3** function has been entered by the dialog box, and the variable **[acre-ft]** has been entered as the input. The numeric calculation **volm3** will be given the output of the**acre_ft_to_m3** function.

Temperature...

This selection opens a dialog box prompting for the **From Unit** and **To Unit** for the conversion.

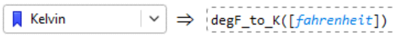


Convert Temperature

From Unit	To Unit
○ Kelvin (K)	○ Kelvin (K)
○ Degrees Celsius (°C)	○ Degrees Celsius (°C)
● Degrees Fahrenheit (°F)	○ Degrees Fahrenheit (°F)

When you have selected the two units, the dialog box disappears and the conversion function you require will have been inserted into the expression at the cursor, and the cursor repositioned so you can enter the source of the value to be converted.

Example



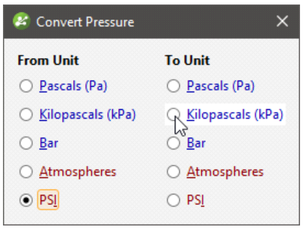
```
Kelvin ⇒ degF_to_K([fahrenheit])
```

Explanation

The **degF_to_K** function has been entered by the dialog box, and the variable **[fahrenheit]** has been entered as the input. The numeric calculation **Kelvin** will be given the output of the**degF_to_K** function.

Pressure...

This selection opens a dialog box prompting for the **From Unit** and **To Unit** for the conversion.

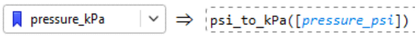


Convert Pressure

From Unit	To Unit
○ Pascals (Pa)	○ Pascals (Pa)
○ Kilopascals (kPa)	○ Kilopascals (kPa)
○ Bar	○ Bar
○ Atmospheres	○ Atmospheres
● PSI	○ PSI

When you have selected the two units, the dialog box disappears and the conversion function you require will have been inserted into the expression at the cursor, and the cursor repositioned so you can enter the source of the value to be converted.

Example



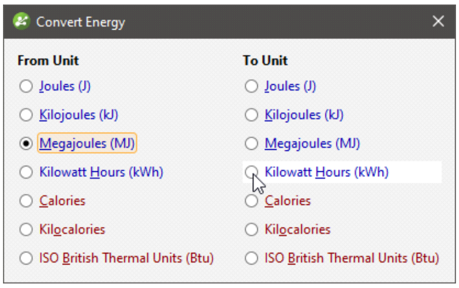
pressure_kPa ⇒ `psi_to_kPa([pressure_psi])`

Explanation

The **psi_to_kPa** function has been entered by the dialog box, and the variable **[pressure_psi]** has been entered as the input. The numeric calculation **pressure_kPa** will be given the output of the**psi_to_kPa** function.

Energy...

This selection opens a dialog box prompting for the **From Unit** and **To Unit** for the conversion.

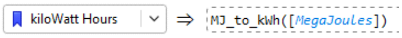


Convert Energy

From Unit	To Unit
○ Joules (J)	○ Joules (J)
○ Kilojoules (kJ)	○ Kilojoules (kJ)
● Megajoules (MJ)	○ Megajoules (MJ)
○ Kilowatt Hours (kWh)	○ Kilowatt Hours (kWh)
○ Calories	○ Calories
○ Kilocalories	○ Kilocalories
○ ISO British Thermal Units (Btu)	○ ISO British Thermal Units (Btu)

When you have selected the two units, the dialog box disappears and the conversion function you require will have been inserted into the expression at the cursor, and the cursor repositioned so you can enter the source of the value to be converted.

Example



kiloWatt Hours ⇒ `MJ_to_kWh([MegaJoules])`

Explanation

The **MJ_to_kWh** function has been entered by the dialog box, and the variable **[MegaJoules]** has been entered as the input. The numeric calculation **kiloWatt Hours** will be given the output of the **MJ_to_kWh** function.