Guide to Calculations and Filters

for Leapfrog Geo version 5.0

Calculations and Filters on Block Models

This guide describes how to use the **Calculations and Filters** editor available for block models in Leapfrog Geo. It is divided into two parts:

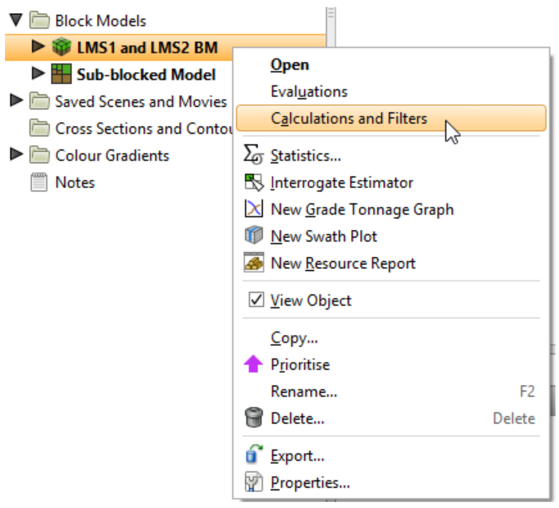
- The first part describes how to define calculations and filters for block models.
- The second part is a Catalogue of Metadata, Syntax and Functions. It describes each of the items in the pinnable **Insert** list and includes intentionally trivial examples to illustrate the use of the item, along with an explanation of the effect of the expression.

> The features described in this guide are only available if you have the Leapfrog Edge extension.

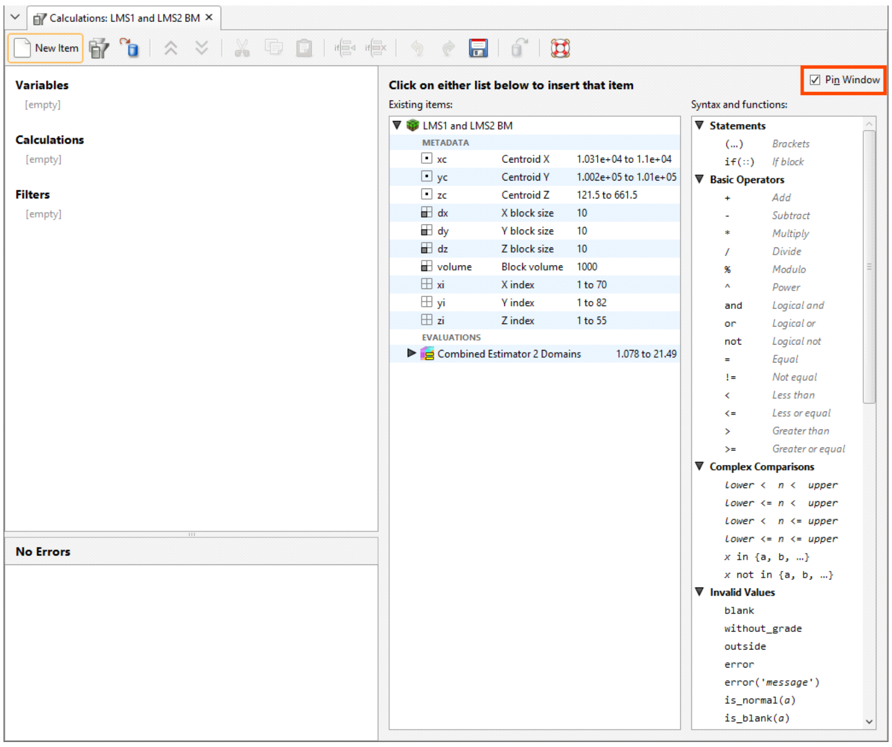
Calculations and Filters use estimators and data to derive new values, resource classifications and more. A powerful and versatile tool, it offers a simple and intuitive editor so you do not need to be an expert programmer to make use of its pure functional language to define calculations. Even complex calculations can be developed with ease.

The Calculations Window

Once you have defined a block model, you can define a set of calculations and filters for it by right-clicking on the block model in the project tree and selecting **Calculations and Filters**:



When the **Calculations** window is first opened, the **Insert** list is pinned to the right-hand side. You can untick the **Pin Window** box to get more screen space:


Calculations: LMS1 and LMS2 BM

New Item

Variables
[empty]

Calculations
[empty]

Filters
[empty]

No Errors

[✓ Pin Window]

Click on either list below to insert that item

Existing items:

LMS1 and LMS2 BM		
METADATA		
xc	Centroid X	1.031e+04 to 1.1e+04
yc	Centroid Y	1.002e+05 to 1.01e+05
zc	Centroid Z	121.5 to 661.5
dx	X block size	10
dy	Y block size	10
dz	Z block size	10
volume	Block volume	1000
xi	X index	1 to 70
yi	Y index	1 to 82
zi	Z index	1 to 55
EVALUATIONS		
Combined Estimator 2 Domains		1.078 to 21.49

Syntax and functions:

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*

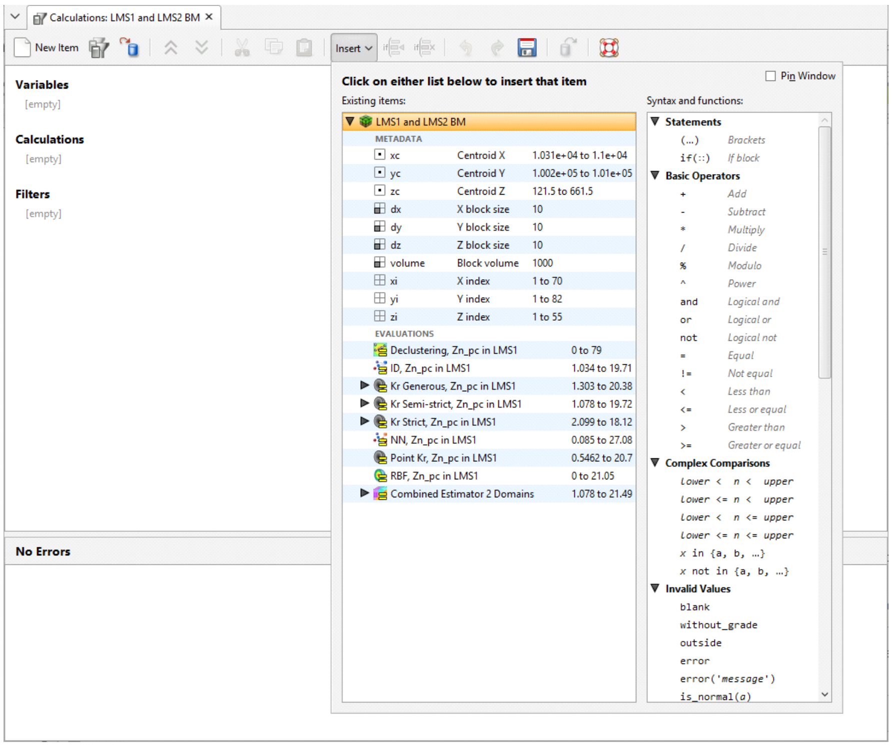
Complex Comparisons

```
lower <  n <  upper
lower <= n <  upper
lower <  n <= upper
lower <= n <= upper
x in {a, b, …}
x not in {a, b, …}
```

Invalid Values

```
blank
without_grade
outside
error
error('message')
is_normal(a)
is_blank(a)
```

You can then select items by clicking on the **Insert** menu:


Calculations: LMS1 and LMS2 BM

New Item

Insert

Variables
[empty]

Calculations
[empty]

Filters
[empty]

No Errors

[Pin Window]

Click on either list below to insert that item

Existing items:

LMS1 and LMS2 BM		
METADATA		
xc	Centroid X	1.031e+04 to 1.1e+04
yc	Centroid Y	1.002e+05 to 1.01e+05
zc	Centroid Z	121.5 to 661.5
dx	X block size	10
dy	Y block size	10
dz	Z block size	10
volume	Block volume	1000
xi	X index	1 to 70
yi	Y index	1 to 82
zi	Z index	1 to 55
EVALUATIONS		
Declustering, Zn_pc in LMS1		0 to 79
ID, Zn_pc in LMS1		1.034 to 19.71
Kr Generous, Zn_pc in LMS1		1.303 to 20.38
Kr Semi-strict, Zn_pc in LMS1		1.078 to 19.72
Kr Strict, Zn_pc in LMS1		2.099 to 18.12
NN, Zn_pc in LMS1		0.085 to 27.08
Point Kr, Zn_pc in LMS1		0.5462 to 20.7
RBF, Zn_pc in LMS1		0 to 21.05
Combined Estimator 2 Domains		1.078 to 21.49

Syntax and functions:

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

```
lower <  n <  upper
lower <= n <  upper
lower <  n <= upper
lower <= n <= upper
x in {a, b, …}
x not in {a, b, …}
```

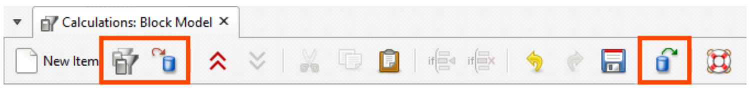
Invalid Values

```
blank
without_grade
outside
error
error('message')
is_normal(a)
```

Items in this list are divided into **Existing items** and **Syntax and functions**. **Existing items** include handy metadata items you can use immediately, e.g. block size and volume measurements. It also includes evaluations that can be selected, such as estimators. **Syntax and functions** contains mathematical operators and other calculation elements, along with special values such as 'blank' and 'outside' and pre-made functions such as unit conversions and *log(n)*.

If the **Calculations** window is docked as a tab, you can tear the tab off to form a stand-alone window. You might do this when arranging your windows so you can see the **Calculations** window at the same time as the scene. The detached window can be docked again by dragging the tab back alongside the other tabs.

The toolbar in the **Calculations** window has three buttons that relate to copying, importing and exporting calculations and filters between block models and projects:



To copy calculations and filters from another block model, click the **Copy** button () and select which model to copy from.

To export calculations and filters, click the **Export** button (). You will be prompted for a file name and location. The information is saved in *.lfcalc format, which is a binary file format. This format cannot be read or written by any other program.

To import calculations, click on the **Import** button () and select a calculations file to import.

The arrow buttons (and) allow you to quickly expand or collapse all calculations and filters. You can also individually expand or collapse a calculation or filter by clicking on the ⇒ symbol next to the calculation or filter name.

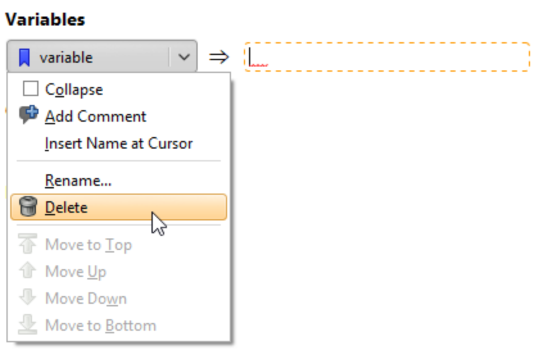
Creating a New Calculation or Filter

Click the **New Item** button to add a calculation to the **Calculations** window:

The **New Item** windows gives you the choice of creating a variable, a numeric calculation, a category calculation, or a filter. Select one of these options, enter a name for the new item and click **OK**.

The item will appear in the list on the left-hand side of the **Calculations** window:

If you want to delete an item, click on it in the list, then select **Delete**:



Variables

A **Variable** is an expression that is given a name to make it simple to refer to the expression in other parts of the **Calculations** window. Note that 'variable' is a homonym/homograph that in a different context has a different meaning; 'variable' is at times used to refer to a mineral such as gold or silver. In the context of calculations and filters, the meaning is aligned with the use of the word in mathematics and scripting languages.

Using a variable in calculations can make the calculations easier to read and understand. It also makes it simple to reuse an expression multiple times in different places. Any time you find the same expression appearing in different calculations, or in different parts of a calculation, split that expression out into a variable. You can even use a variable to represent a constant value, such as a particular density measure. You can use it to hold a value you want to change as you experiment. If the value is used in multiple places in other expressions and calculations, using a variable in those places means you only need to change the value in one place instead of many.

Numeric Calculations

A **Numeric calculation** is an expression that evaluates to a number, or a number for each block evaluation used as input. A numeric calculation can be viewed in the 3D scene as a block model.

Category Calculations

A **Category calculation** is an expression that evaluates to text, usually used to label a category or classification such as a lithology or grade description. If a block evaluation is used as input, each block will be assigned a text label to categorise it according to the evaluation value. A category calculation can be viewed in the 3D scene as a block model.

Filters

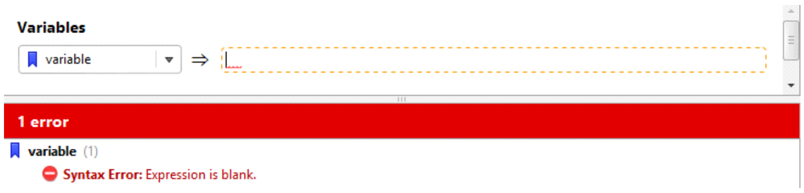
A **Filter** is an expression that limits the data to specified constraints. When viewing a block model in the scene view, filters can be selected in the properties panel's **Query filter** option to constrain which blocks are displayed.

Understanding Errors Reported in the Errors Pane

Items in the **Existing items** list or in the **Syntax and functions** list can be selected and added to the **Calculations** pane. While a calculation or filter is incomplete, the **Error** pane will identify what is needed to make the calculation valid.

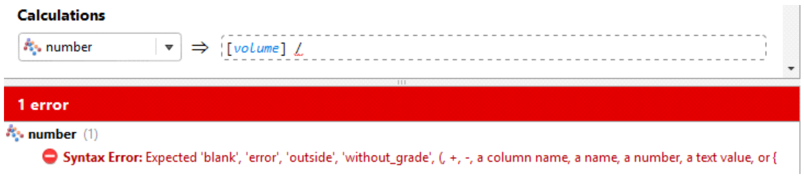
The errors pane will report when the syntax of an expression cannot be validly interpreted. It cannot report when the expression is not correctly structured to do what was intended.

While you are in the process of constructing an expression, errors will be reported when the incomplete expression cannot be validly interpreted:



Syntax Error: Expression is blank.

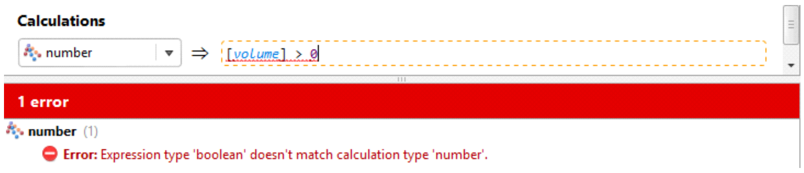
The messages provide the reason the expression is not valid or complete, but it cannot tell you how to complete or correct the equation.



[volume] /

Syntax Error: Expected 'blank', 'error', 'outside', 'without_grade', (, +, -, a column name, a name, a number, a text value, or {

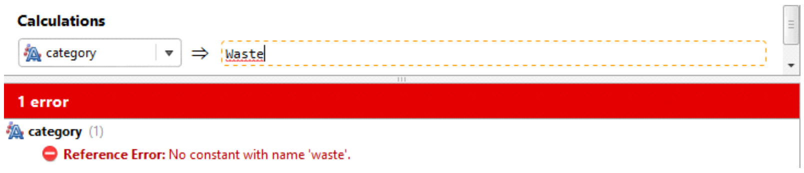
Other errors will be displayed when the type of result doesn't match the sort of calculation selected when it was created. A numeric calculation needs to produce numbers, a category calculation needs to produce text results, and filters need to produce true or false (boolean) results.



[volume] > 0

Error: Expression type 'boolean' doesn't match calculation type 'number'.

Text needs to be identified by enclosing it with quotation marks, so it is not mistaken by the calculation engine as some sort of unspecified constant.



Waste

Reference Error: No constant with name 'waste'.

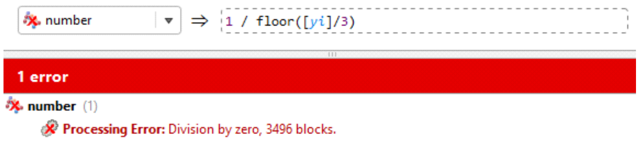
A processing error indicates a problem executing part of the calculation, for at least one of the blocks. The function may be syntactically correct, but the calculation cannot be performed for one or more blocks for some other reason. In the example below, **number** will not be able to be calculated if the Y index is ever below 3, as that will result in a divide by zero error. Because the Y index ranges from 1 upwards, this does indeed cause a processing error. Not all processing errors will be divide by zero errors. The list of problems that will be reported as processing errors includes:

- divide by zero
- log of zero or a negative number
- log base of 1
- raising a negative number to a fractional power

- sqrt of a negative number
- numeric overflow, producing a number outside the range -1e308 to 1e308
- a null value (blank, outside, without_grade) in a set
- multiple instances of the same value in a set
- referencing a column that has been deleted
- referencing a column that contains processing errors

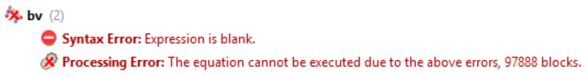
Some processing errors will report that the problem is due to 'above errors', referring to errors higher in the errors list.

Processing errors will include a count of the number of affected blocks.



```
number ⇒ 1 / floor([yi]/3)
1 error
number (1)
    Processing Error: Division by zero, 3496 blocks.
```

You may have more than one error shown. As a rule of thumb, address the errors at the top first, as this corrective action may address the subsequent error.

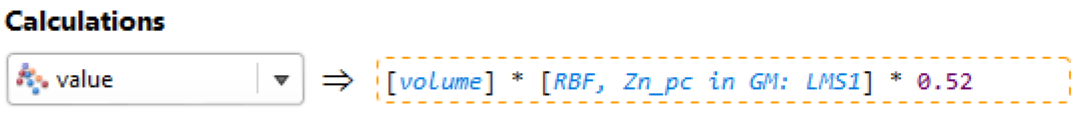


```
bv (2)
    Syntax Error: Expression is blank.
    Processing Error: The equation cannot be executed due to the above errors, 97888 blocks.
```

Building Calculations and Filters

Items in the **Existing items** list or in the **Syntax and functions** list can be selected and added to the **Calculations** pane. Wherever a dotted box appears in the **Calculations** pane, you can add items and operators. Select an insertion point by clicking the dotted box where the item is required. You will see that the colour of the dotted box turns from black to orange to indicate that it is the currently selected entry box. Insert an item from one of the lists by clicking on it.

Select additional operators, constants, evaluations or other items as required, or type in values:



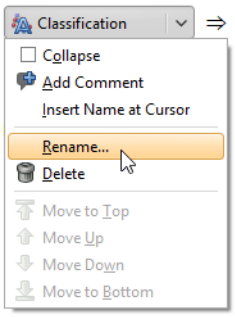
```
Calculations
value ⇒ [volume] * [RBF, Zn_pc in GM: LMS1] * 0.52
```

Some items you insert include placeholders that need to be replaced. For example, here, a value needs to be entered at the cursor position and the terms ‘lower’ and ‘upper’ need to be replaced with other items:

The if (::) conditional statement can have additional rows added to it. Insert the cursor where you want to insert a row and then click the **Add Row** button (📋) and a row will be added above the selected line:

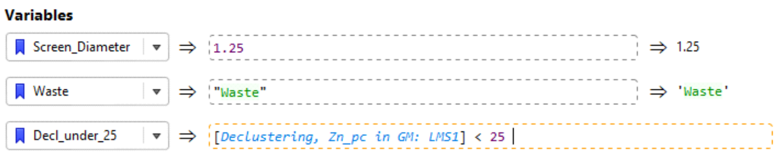
You can hold the Shift key to add the row below the cursor. To delete the selected row, click the **Delete Row** button (📋).

A calculation or filter can be renamed after it has been created. Click on its name and select **Rename** from the menu that appears:



Examples of Variables

Variables can produce numeric, boolean, or text results. You may create a variable because using a variable in other expressions can make them easier to read and understand, or it can let you change a value in just one place, even though it is used in many expressions.

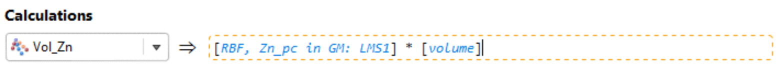

Variables

```
Screen_Diameter ⇒ 1.25 ⇒ 1.25
Waste ⇒ "Waste" ⇒ 'Waste'
Decl_under_25 ⇒ [Declustering, Zn_pc in GM: LMS1] < 25
```

While producing results similar to numeric calculations, category calculations or filters, variables cannot be viewed on the block model in the 3D scene.

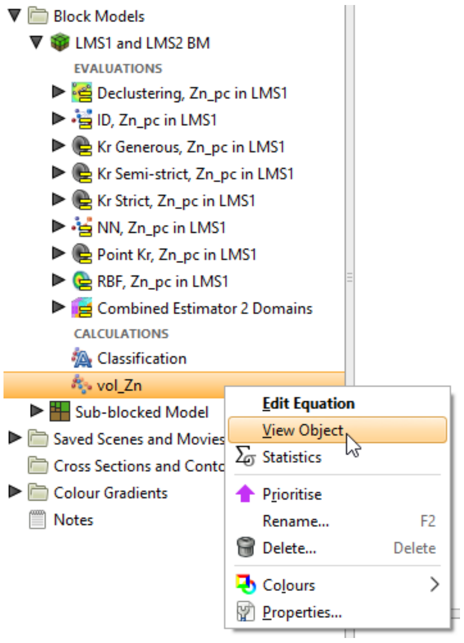
Example of a Numeric Calculation

Numeric calculations must produce numeric results, or an error will be reported.


Calculations

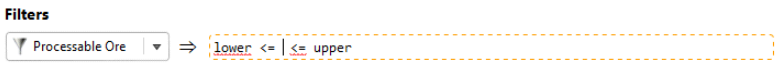
```
Vol_Zn ⇒ [RBF, Zn_pc in GM: LMS1] * [volume]
```

The results of the numeric calculation for the block model can be viewed in the 3D scene by right-clicking on the numeric calculation entry in the project tree and selecting **View Object**.



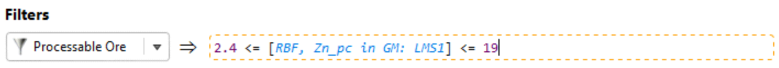
Example of a Filter

In this example, one of the selectors from the **Complex Comparisons** options has chosen from the insert list. The value that will be tested is to be added at the insertion point between the comparators, and the placeholder words 'lower' and 'upper' need to be replaced with constants.



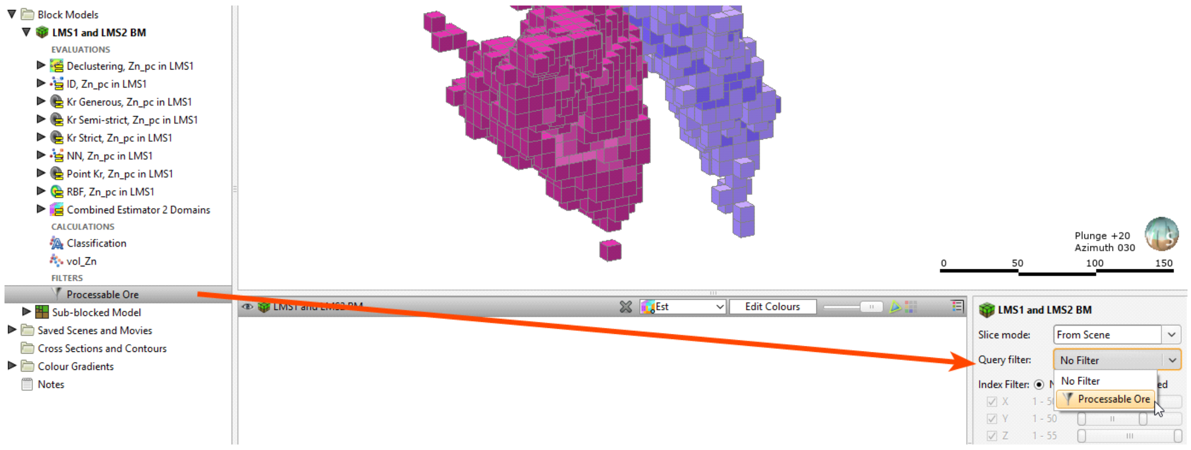
```
lower <= | <= upper
```

Here the RBF estimator has been specified as the value that will be tested, and lower and upper limits for the processable ore have been specified.



```
2.4 <= [RBF, Zn_pc in GM: LMS1] <= 19
```

When you select **View Object** for the filter in the project tree, the block model will be displayed with the **Query filter** set to the selected filter in the shape properties panel.

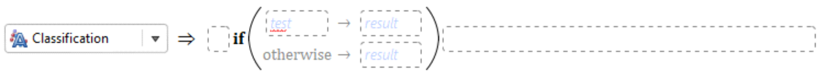


LMS1 and LMS2 BM
Slice mode: From Scene
Query filter: No Filter
Index Filter:
X 1 - 5
Y 1 - 50
Z 1 - 55

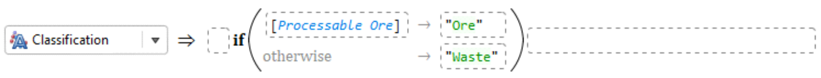
Example of a Category Calculation

This example use a category calculation to classify ore into one of several categories using the if (::) conditional statement.

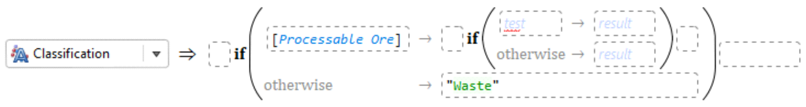
The if(::) conditional statement contains two lines by default. Note the guiding cues within the dotted entry boxes. The first line describes the test to be performed that, if it evaluates as true, will produce a specified result. The ‘otherwise’ condition on the second line captures all the tested conditions that did not evaluate as true in the test on the first line, and these all produce an alternate result:


```
Classification ⇒ if( test      → result
                     otherwise → result )
```

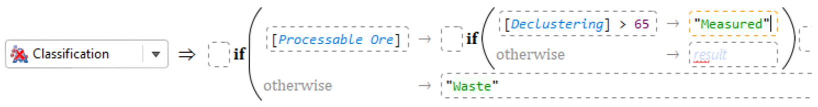
In this case, all the values outside of the range of the filter we have created are to be considered waste. We add the filter as the ‘test’ condition, and specify the label ‘Ore’ as the first result, and use the label ‘Waste’ in the bottom line for the ‘otherwise’ condition. So the labels are not mistaken by the calculation engine as some sort of unspecified constant, make sure labels are identified using quotation marks:


```
Classification ⇒ if( [Processable Ore] → "Ore"
                     otherwise         → "Waste" )
```

To extend this trivial example to make it more practical, we can nest if(::) conditional statements. Replace the ‘Ore’ label with a new if(::) item:

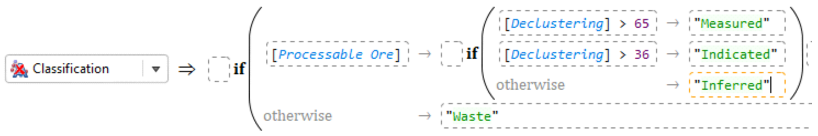

```
Classification ⇒ if( [Processable Ore] → if( test      → result
                                             otherwise → result )
                     otherwise         → "Waste" )
```

Now if values pass the filter in the first test, they then get to be sorted using the second if(::) block. This time we will test a declustering object from the sample geometry. If the Declustering value is more than 65, we will classify this as 'Measured':



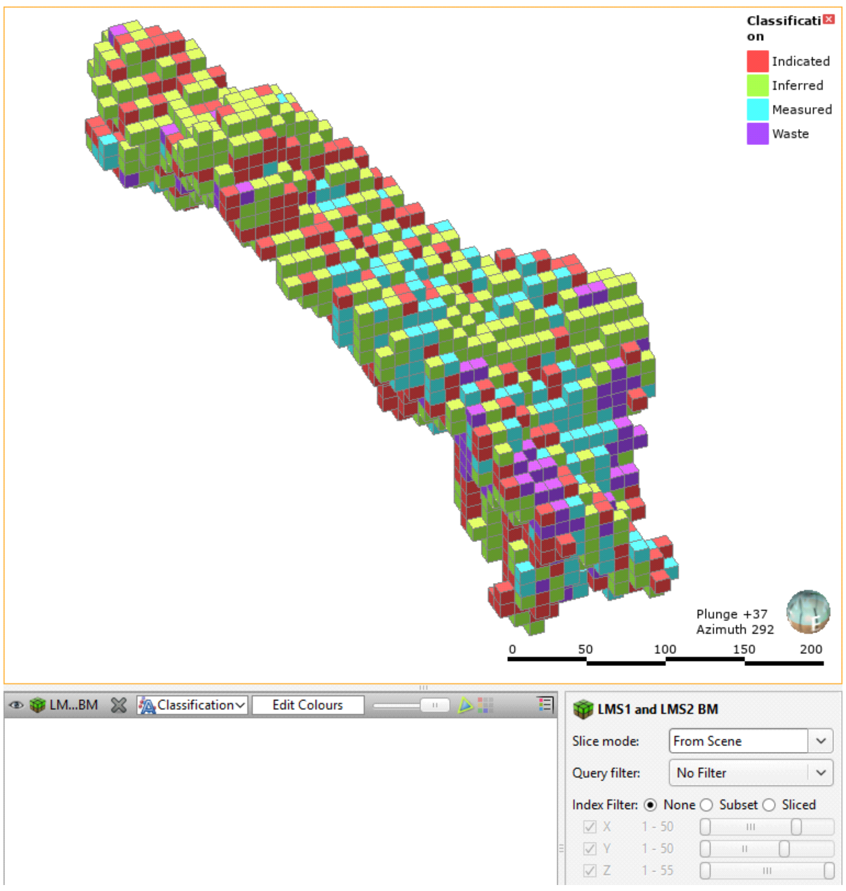
```
Classification ⇒ if ( [Processable Ore] → if ( [Declustering] > 65 → "Measured"
                                                otherwise           → result )
                      otherwise          → "Waste" )
```

Shift-click the **Add Row** button (), and a new conditional row is added below the current one, where we can specify that a Declustering value greater than 36 will be classified as 'Indicated'. Only values between 65 and 36 will be classified as 'Indicated' as the first line will have already classified all the values over 65 as 'Measured'. This leaves the 'otherwise' row in the nested if(::) block to catch data density values below 36, which are labelled 'Inferred':



```
Classification ⇒ if ( [Processable Ore] → if ( [Declustering] > 65 → "Measured"
                                                [Declustering] > 36 → "Indicated"
                                                otherwise           → "Inferred" )
                      otherwise          → "Waste" )
```

The new Classification calculation we have added under the block model can be added to the scene view.



Classification
- 🟥 Indicated
- 🟩 Inferred
- 🟦 Measured
- 🟪 Waste

LMS1 and LMS2 BM

Slice mode: From Scene

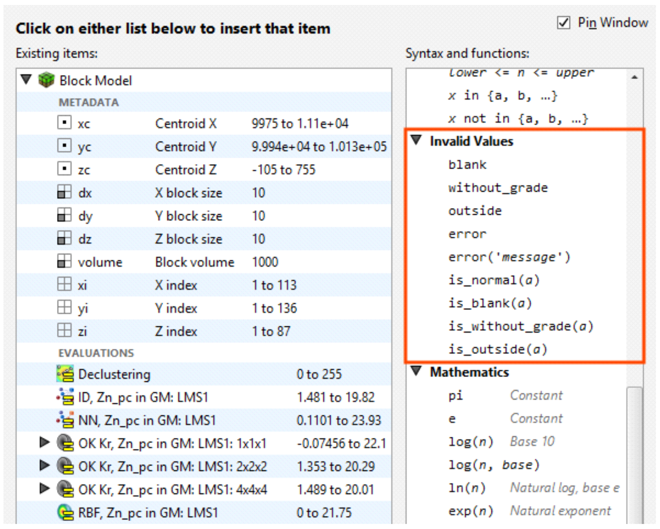
Query filter: No Filter

Index Filter: [● None] ○ Subset ○ Sliced

☑ X	1 - 50
☑ Y	1 - 50
☑ Z	1 - 55

Null Values

Leapfrog Geo can differentiate between different types of invalid or null values, which are shown in the **Calculations** window:



Click on either list below to insert that item

☑ Pin Window

Existing items:

Block Model		
METADATA		
xc	Centroid X	9975 to 1.11e+04
yc	Centroid Y	9.994e+04 to 1.013e+05
zc	Centroid Z	-105 to 755
dx	X block size	10
dy	Y block size	10
dz	Z block size	10
volume	Block volume	1000
xi	X index	1 to 113
yi	Y index	1 to 136
zi	Z index	1 to 87
EVALUATIONS		
Declustering		0 to 255
ID, Zn_pc in GM: LMS1		1.481 to 19.82
NN, Zn_pc in GM: LMS1		0.1101 to 23.93
OK Kr, Zn_pc in GM: LMS1: 1x1x1		-0.07456 to 22.1
OK Kr, Zn_pc in GM: LMS1: 2x2x2		1.353 to 20.29
OK Kr, Zn_pc in GM: LMS1: 4x4x4		1.489 to 20.01
RBF, Zn_pc in GM: LMS1		0 to 21.75

Syntax and functions:

```
lower <= n <= upper
x in {a, b, …}
x not in {a, b, …}
```

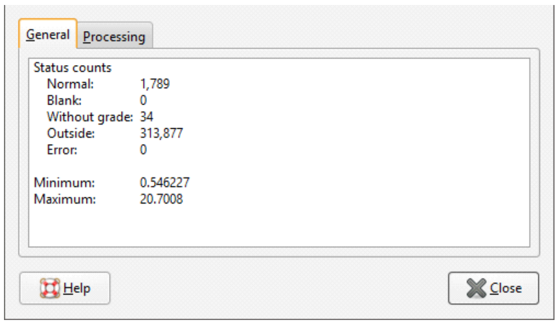
Invalid Values

```
blank
without_grade
outside
error
error('message')
is_normal(a)
is_blank(a)
is_without_grade(a)
is_outside(a)
```

Mathematics

`pi`	*Constant*
`e`	*Constant*
`log(n)`	*Base 10*
`log(n, base)`	
`ln(n)`	*Natural log, base e*
`exp(n)`	*Natural exponent*

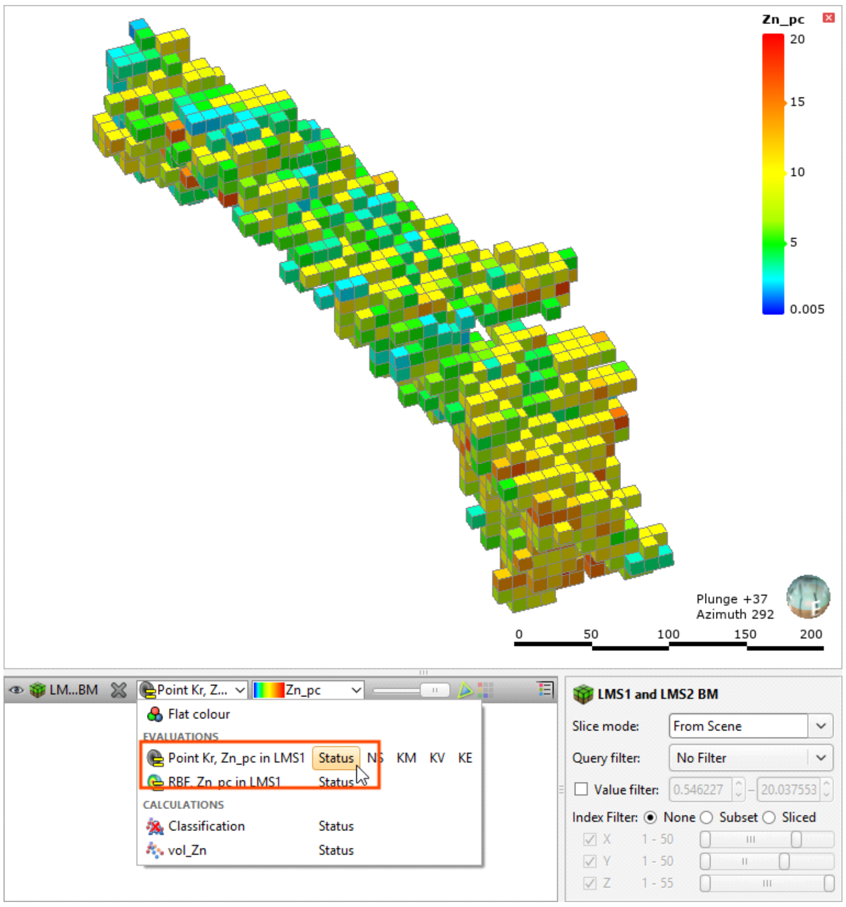
You can see how many blocks of each normal or invalid status occur in an evaluation by right-clicking on the evaluation in the project tree and selecting **Properties**:



```
Status counts
  Normal:        1,789
  Blank:         0
  Without grade: 34
  Outside:       313,877
  Error:         0

Minimum:         0.546227
Maximum:         20.7008
```

When a block model is displayed in the scene, you can choose between displaying evaluation values or the block status. Highlighted below is a Kriging evaluation, shown displayed on the block model. Next to the Kriging evaluation in the dropdown list is a **Status** option:



LMS1 and LMS2 BM

Slice mode: From Scene

Query filter: No Filter

Value filter: 0.546227 – 20.037553

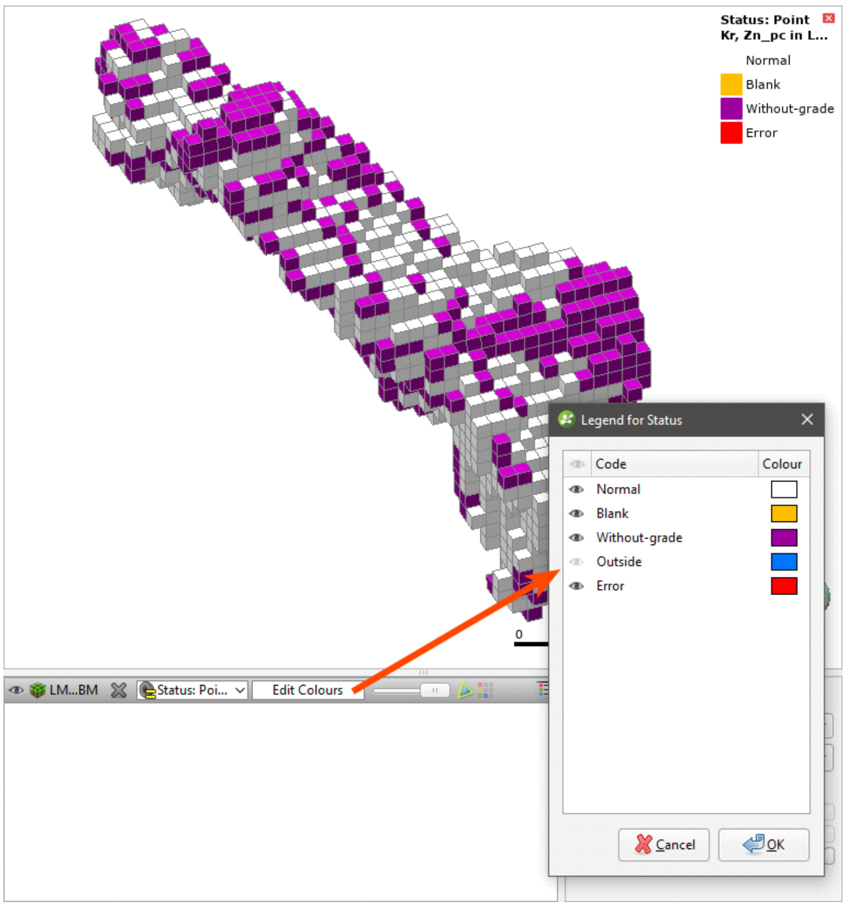
Index Filter: [✓ None] Subset Sliced

X 1 - 50

Y 1 - 50

Z 1 - 55

When the status option is selected, the model is displayed by block status. Click **Edit Colours** to change what status values are displayed.



Status: Point Kr, Zn_pc in L...

⬜	Normal
🟨	Blank
🟪	Without-grade
🟥	Error

LM...BM · Status: Poi... · Edit Colours

Legend for Status

Code	Colour
Normal	⬜
Blank	🟨
Without-grade	🟪
Outside	🟦
Error	🟥

Cancel · OK

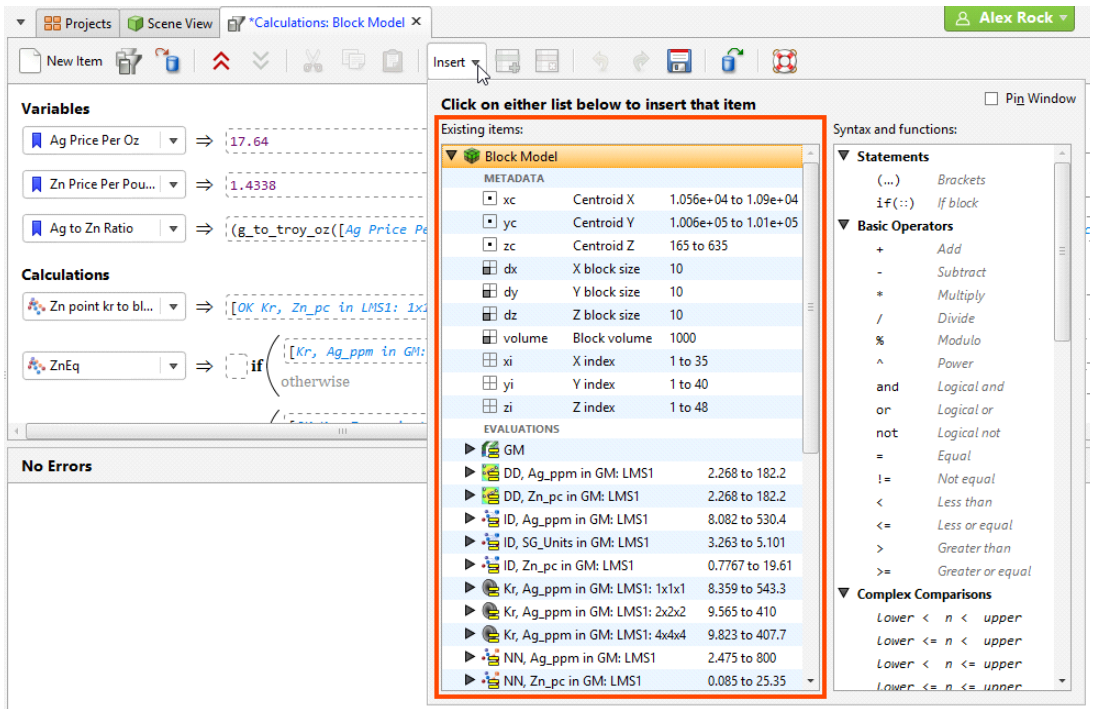
With block model statistics, you can view statistics for all evaluations and calculations made on a block model or sub-blocked model. Statistics can be broken down into categories and organised by numeric evaluations and calculations.

Catalogue of Metadata, Syntax and Functions

This catalogue details each of the items in the pinnable **Insert** list. 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:



Variables

Ag Price Per Oz	⇒ 17.64
Zn Price Per Pou...	⇒ 1.4338
Ag to Zn Ratio	⇒ (g_to_troy_oz([Ag Price Pe

Calculations

Zn point kr to bl...	⇒ [OK Kr, Zn_pc in LMS1: 1x1
ZnEq	⇒ if ([Kr, Ag_ppm in GM: … otherwise

No Errors

Click on either list below to insert that item

Existing items:

Block Model

METADATA

xc	Centroid X	1.056e+04 to 1.09e+04
yc	Centroid Y	1.006e+05 to 1.01e+05
zc	Centroid Z	165 to 635
dx	X block size	10
dy	Y block size	10
dz	Z block size	10
volume	Block volume	1000
xi	X index	1 to 35
yi	Y index	1 to 40
zi	Z index	1 to 48

EVALUATIONS

GM	
DD, Ag_ppm in GM: LMS1	2.268 to 182.2
DD, Zn_pc in GM: LMS1	2.268 to 182.2
ID, Ag_ppm in GM: LMS1	8.082 to 530.4
ID, SG_Units in GM: LMS1	3.263 to 5.101
ID, Zn_pc in GM: LMS1	0.7767 to 19.61
Kr, Ag_ppm in GM: LMS1: 1x1x1	8.359 to 543.3
Kr, Ag_ppm in GM: LMS1: 2x2x2	9.565 to 410
Kr, Ag_ppm in GM: LMS1: 4x4x4	9.823 to 407.7
NN, Ag_ppm in GM: LMS1	2.475 to 800
NN, Zn_pc in GM: LMS1	0.085 to 25.35

Syntax and functions:

Statements

(...)	*Brackets*
if(::)	*If block*

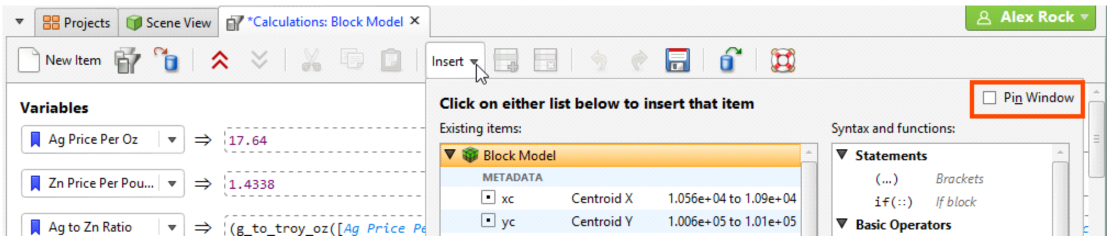
Basic Operators

+	*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

```
lower <  n <  upper
lower <= n <  upper
lower <  n <= upper
lower <= n <= upper
```

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



Variables

Ag Price Per Oz	⇒ 17.64
Zn Price Per Pou...	⇒ 1.4338
Ag to Zn Ratio	⇒ (g_to_troy_oz([Ag Price Pe

Click on either list below to insert that item

Existing items:

Block Model

METADATA

xc	Centroid X	1.056e+04 to 1.09e+04
yc	Centroid Y	1.006e+05 to 1.01e+05

Syntax and functions:

Statements

(...)	*Brackets*
if(::)	*If block*

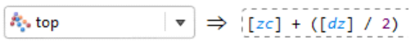
Basic Operators

Metadata

xc, yc and zc

These three metadata items are the variables for locating the centroid of the 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.

Example



top ⇒ `[zc] + ([dz] / 2)`

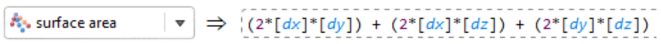
Explanation

The numeric calculation top will be assigned the value of the location of [zc] the altitude of the 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.

Example



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

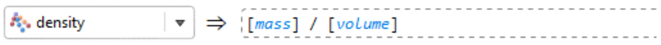
Explanation

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.

Example



density ⇒ `[mass] / [volume]`

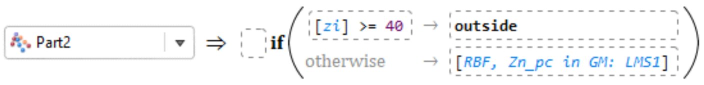
Explanation

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 the 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.

Example



Explanation

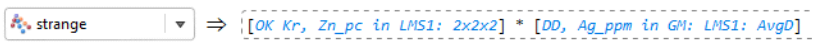
The numeric calculation Part2 will be assigned the value of the RBF estimation, unless [zi] the Z index of the block is greater than or equal to 40, in which case the block status will be set to the invalid value outside.

Evaluations

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

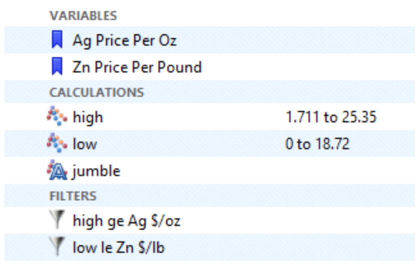

strange
⇒ [OK Kr, Zn_pc in LMS1: 2x2x2] * [DD, Ag_ppm in GM: LMS1: AvgD]

Explanation

The numeric calculation strange is defined by [OK Kr, Zn_pc in LMS1: 2x2x2] the product of the Kriging estimate and [DD, Ag_ppm in GM: LMS1: AvgD] the average distance to sample attribute for the Data Density estimator. Each block in the block model will have its own values for the Kriging estimate and the Data Density AvgD attribute, and this calculation uses those values to create a new value named strange for each block , using the formula above.

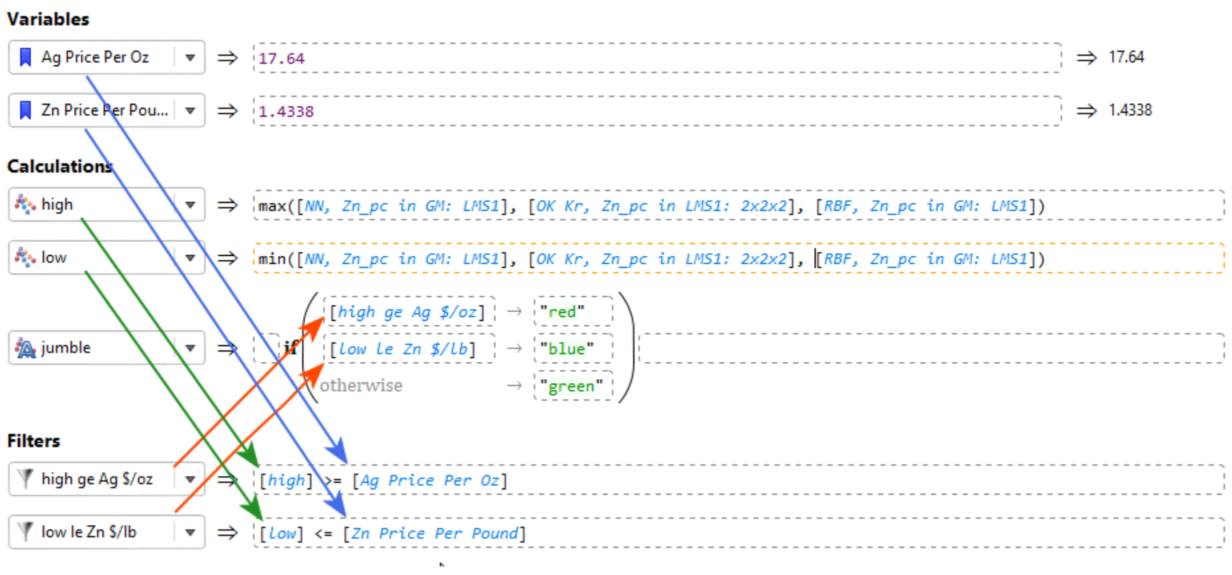
Variables, Calculations and Filters

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	
Ag Price Per Oz	
Zn Price Per Pound	
CALCULATIONS	
high	1.711 to 25.35
low	0 to 18.72
jumble	
FILTERS	
high ge Ag $/oz	
low le Zn $/lb	

Example

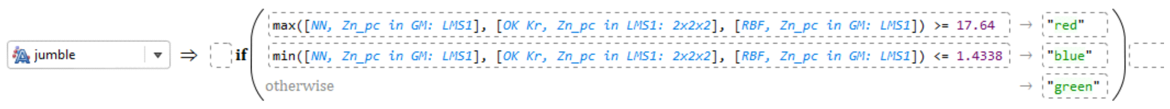


```
Variables
Ag Price Per Oz   ⇒ 17.64   ⇒ 17.64
Zn Price Per Pou... ⇒ 1.4338   ⇒ 1.4338
Calculations
high    ⇒ max([NN, Zn_pc in GM: LMS1], [OK Kr, Zn_pc in LMS1: 2x2x2], [RBF, Zn_pc in GM: LMS1])
low     ⇒ min([NN, Zn_pc in GM: LMS1], [OK Kr, Zn_pc in LMS1: 2x2x2], [RBF, Zn_pc in GM: LMS1])
jumble  ⇒ if ( [high ge Ag $/oz]  → "red"
               [low le Zn $/lb]   → "blue"
               otherwise          → "green" )
Filters
high ge Ag $/oz  ⇒ [high] >= [Ag Price Per Oz]
low le Zn $/lb   ⇒ [low] <= [Zn Price Per Pound]
```

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; there is nothing useful about the calculation jumble.

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. As a counterpoint, compare the following calculation that does the same jumble calculation above, but it is more difficult to interpret because everything is thrown in together:

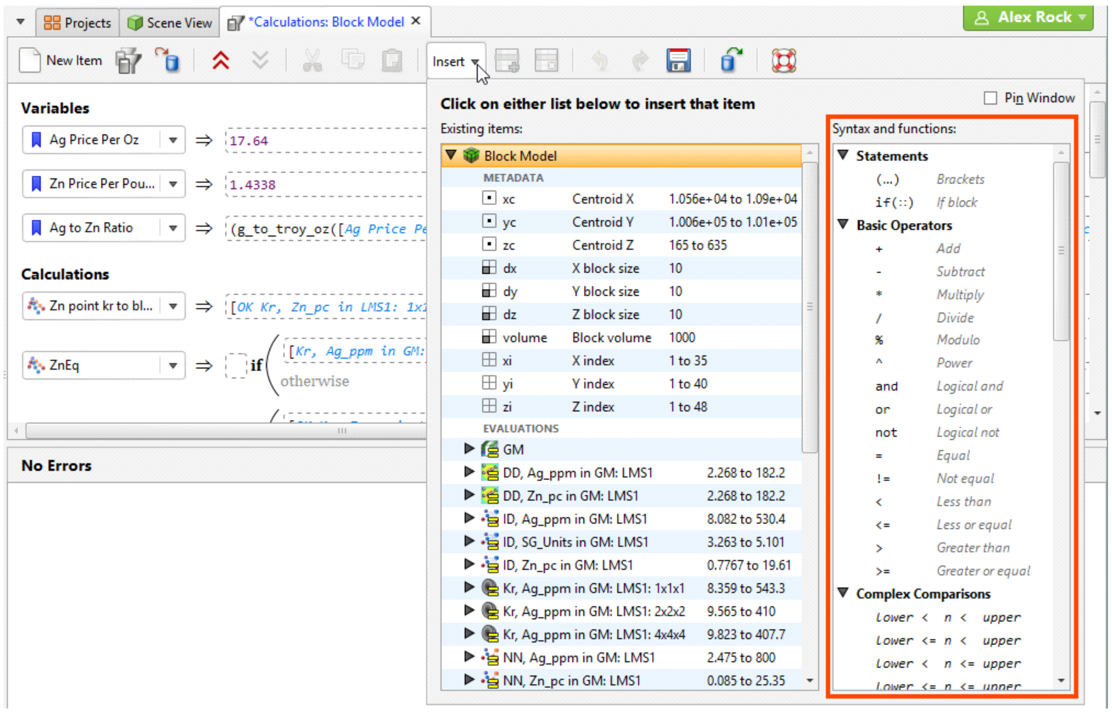


```
jumble ⇒ if ( max([NN, Zn_pc in GM: LMS1], [OK Kr, Zn_pc in LMS1: 2x2x2], [RBF, Zn_pc in GM: LMS1]) >= 17.64  → "red"
              min([NN, Zn_pc in GM: LMS1], [OK Kr, Zn_pc in LMS1: 2x2x2], [RBF, Zn_pc in GM: LMS1]) <= 1.4338 → "blue"
              otherwise                                                                                      → "green" )
```

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.



Variables

```
Ag Price Per Oz     ⇒ 17.64
Zn Price Per Pou... ⇒ 1.4338
Ag to Zn Ratio      ⇒ (g_to_troy_oz([Ag Price Pe
```

Calculations

```
Zn point kr to bl... ⇒ [OK Kr, Zn_pc in LMS1: 1x1
ZnEq                 ⇒ if( [Kr, Ag_ppm in GM:
                           otherwise
```

No Errors

Click on either list below to insert that item

Pin Window

Existing items:

Block Model

METADATA		
xc	Centroid X	1.056e+04 to 1.09e+04
yc	Centroid Y	1.006e+05 to 1.01e+05
zc	Centroid Z	165 to 635
dx	X block size	10
dy	Y block size	10
dz	Z block size	10
volume	Block volume	1000
xi	X index	1 to 35
yi	Y index	1 to 40
zi	Z index	1 to 48
EVALUATIONS		
GM		
DD, Ag_ppm in GM: LMS1		2.268 to 182.2
DD, Zn_pc in GM: LMS1		2.268 to 182.2
ID, Ag_ppm in GM: LMS1		8.082 to 530.4
ID, SG_Units in GM: LMS1		3.263 to 5.101
ID, Zn_pc in GM: LMS1		0.7767 to 19.61
Kr, Ag_ppm in GM: LMS1: 1x1x1		8.359 to 543.3
Kr, Ag_ppm in GM: LMS1: 2x2x2		9.565 to 410
Kr, Ag_ppm in GM: LMS1: 4x4x4		9.823 to 407.7
NN, Ag_ppm in GM: LMS1		2.475 to 800
NN, Zn_pc in GM: LMS1		0.085 to 25.35

Syntax and functions:

Statements

(...)	*Brackets*
if(::)	*If block*

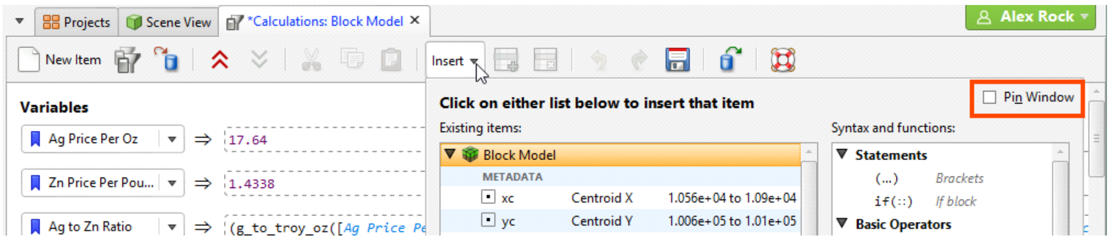
Basic Operators

+	*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

```
lower <  n <  upper
lower <= n <  upper
lower <  n <= upper
lower <= n <= upper
```

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



Variables

Ag Price Per Oz ⇒ 17.64
Zn Price Per Pou... ⇒ 1.4338
Ag to Zn Ratio ⇒ (g_to_troy_oz([Ag Price Pe

Click on either list below to insert that item

Pin Window

Existing items:

Block Model
METADATA
xc Centroid X 1.056e+04 to 1.09e+04
yc Centroid Y 1.006e+05 to 1.01e+05

Syntax and functions:

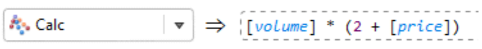
Statements
(...) Brackets
if(::) If block
Basic Operators

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



```
Calc ⇒ [volume] * (2 + [price])
```

Explanation

2 + [price] will be calculated prior to [volume] * the result of the expression in the brackets.

if(::) *If block*

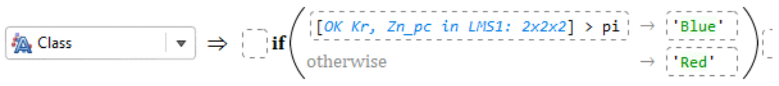
The **If block** is used for conditional logic. This allows multiple pathways to results depending on selected conditions, or categorisation based on values.

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_grade' 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_grade, 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_grade > blank.

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

Example

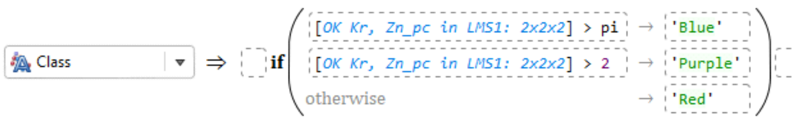


```
Class ⇒ if( [OK Kr, Zn_pc in LMS1: 2x2x2] > pi → 'Blue'
            otherwise                          → 'Red' )
```

Explanation

[OK Kr, Zn_pc in LMS1: 2x2x2] is the automatically generated variable name for the block estimate from a Kriging estimate of Zn_pc (Zinc, percentage) values in a LMS1 domain with 2x2x2 discretisation. As the **if(::) If block** calculation is run for each of the blocks in the model, the estimate for that block replaces this variable name in the expression. If the value is greater than the value of the constant pi, the result 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.

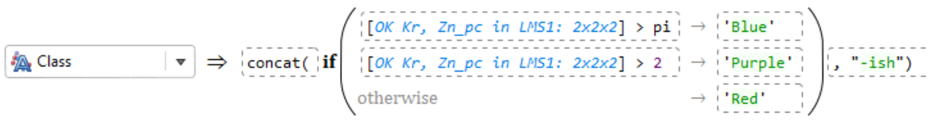


```
Class ⇒ if( [OK Kr, Zn_pc in LMS1: 2x2x2] > pi → 'Blue'
            [OK Kr, Zn_pc in LMS1: 2x2x2] > 2  → 'Purple'
            otherwise                          → 'Red' )
```

Explanation

Note the addition of the line [OK Kr, Zn_pc in LMS1: 2x2x2] > 2 ⟶ 'Purple'. This line can be interpreted to mean: if the value estimated for the block is less than or equal to pi, but greater than 2, 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 [OK Kr, Zn_pc in LMS1: 2x2x2] > 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.



```
Class ⇒ concat( if( [OK Kr, Zn_pc in LMS1: 2x2x2] > pi → 'Blue'
                    [OK Kr, Zn_pc in LMS1: 2x2x2] > 2  → 'Purple'
                    otherwise                          → 'Red' ) , "-ish")
```

Explanation

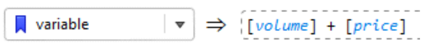
The earlier conditional classification has now been embedded within a concatenation function, forming the first part of a two-part string concatenation. The concatenation function is adding the text string "-ish" to whatever is produced by the **if(::) If block**. Thus, if the value of the block being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

Basic Operators

+ Add

An arithmetic addition operation.

Example



```
variable ⇒ [volume] + [price]
```

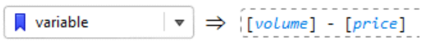
Explanation

variable is assigned the value of [volume] plus [price].

- Subtract

An arithmetic subtraction operation.

Example


variable ⇒ [volume] - [price]

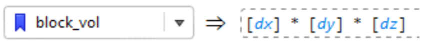
Explanation

variable is assigned the value of [volume] minus [price].

*** Multiply**

An arithmetic multiplication operation.

Example


block_vol ⇒ [dx] * [dy] * [dz]

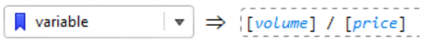
Explanation

block_vol is assigned the value of [dx] times [dy] times [dz]. Note: you do not need to calculate the block volume like this, as the volume metadata item already exists under **Existing items**.

/ Divide

An arithmetic division operation.

Example


variable ⇒ [volume] / [price]

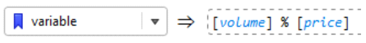
Explanation

variable is assigned the value of [volume] divided by [price].

% Modulo

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

Example


variable ⇒ [volume] % [price]

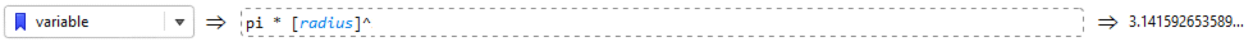
Explanation

variable is assigned the value of [volume] modulo [price], or in other words, [volume] is divided by [price] to produce an integer quotient , the number of times [price] goes into [volume], and the 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



```
variable ⇒ pi * [radius]^ ⇒ 3.141592653589...
```

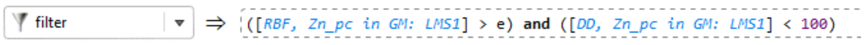
Explanation

variable is assigned the value of pi * [radius]^2 (because ^2 is interpreted as ‘exponent 2’ or 'squared'). Because [radius] happened to be defined as equalling 1, the result of pi * 1^2 equals pi, as can be seen from the result at the end of the expression ⇒ 3.141592653589...

and *Logical and*

A logical and operation.

Example



```
filter ⇒ ([RBF, Zn_pc in GM: LMS1] > e) and ([DD, Zn_pc in GM: LMS1] < 100)
```

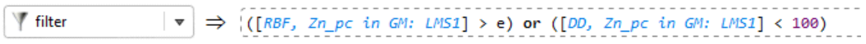
Explanation

filter will be true if the RBF estimate for the block is greater than e AND the data density function is less than 100, but it will be false otherwise.

or *Logical or*

A logical or operation.

Example



```
filter ⇒ ([RBF, Zn_pc in GM: LMS1] > e) or ([DD, Zn_pc in GM: LMS1] < 100)
```

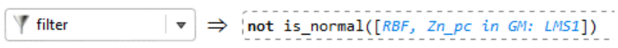
Explanation

filter will be true if the RBF estimate for the block is greater than e OR the data density function is less than 100, but it will be false otherwise.

not *Logical not*

A logical not operation.

Example


filter ⇒ `not is_normal([RBF, Zn_pc in GM: LMS1])`

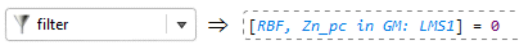
Explanation

filter will be true if the result of the function is_normal using the block RBF estimate value as input returns false; filter will be false if the result of the is_normal function returns true. The logical operator not inverts the logical expression that follows the operator.

= *Equal*

A logical equality operator.

Example


filter ⇒ `[RBF, Zn_pc in GM: LMS1] = 0`

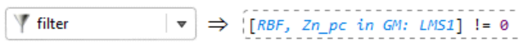
Explanation

filter will be true when the value of the RBF estimate for the block is equal to 0, and will be false for all other values.

!= *Not equal*

A logical not-equal operator.

Example


filter ⇒ `[RBF, Zn_pc in GM: LMS1] != 0`

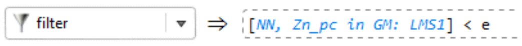
Explanation

filter will be true when the value of the RBF estimate for the block is not equal to 0, and will be false when it does equal 0.

< *Less than*

A logical less-than operator.

Example



filter ⇒ [NN, Zn_pc in GM: LMS1] < e

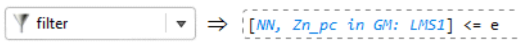
Explanation

filter will be true when the value from the Nearest Neighbour estimate for the block is less than e, and false otherwise.

< *Less or equal*

A logical less-than-or-equals operator.

Example



filter ⇒ [NN, Zn_pc in GM: LMS1] <= e

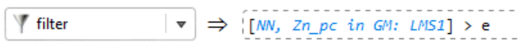
Explanation

filter will be true when the value from the Nearest Neighbour estimate for the block is less than or equal to e, and false otherwise.

< *Greater than*

A logical greater-than operator.

Example



filter ⇒ [NN, Zn_pc in GM: LMS1] > e

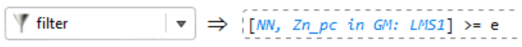
Explanation

filter will be true when the value from the Nearest Neighbour estimate for the block is greater than e, and false otherwise.

< *Greater or equal*

A logical greater-than-or-equals operator.

Example



filter ⇒ [NN, Zn_pc in GM: LMS1] >= e

Explanation

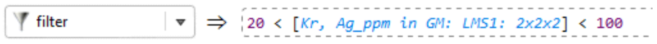
filter will be true when the value from the Nearest Neighbour estimate for the block is greater or equal to e, and false otherwise.

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 ⇒ 20 < [Kr, Ag_ppm in GM: LMS1: 2x2x2] < 100

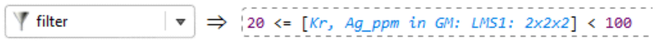
Explanation

filter will be true when the Kriging estimation for the block is between the values of 20 and 100 (but not equalling 20 or 100); 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 ⇒ 20 <= [Kr, Ag_ppm in GM: LMS1: 2x2x2] < 100

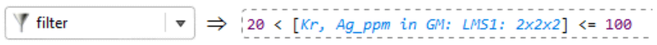
Explanation

filter will be true when the Kriging estimation for the block is equal to 20 or between the values of 20 and 100 (but not equalling 100); 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 ⇒ 20 < [Kr, Ag_ppm in GM: LMS1: 2x2x2] <= 100

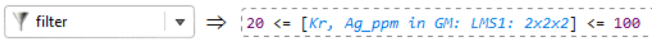
Explanation

filter will be true when the Kriging estimation for the block is between the values of 20 and 100 or equal to 100 (but not equal to 20); 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 ⇒ 20 <= [Kr, Ag_ppm in GM: LMS1: 2x2x2] <= 100
```

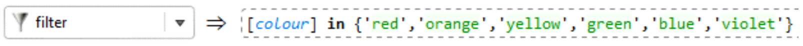
Explanation

filter will be true when the Kriging estimation for the block is greater than or equal to 20 and less than or equal to 100; 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'}
```

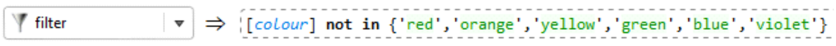
Explanation

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

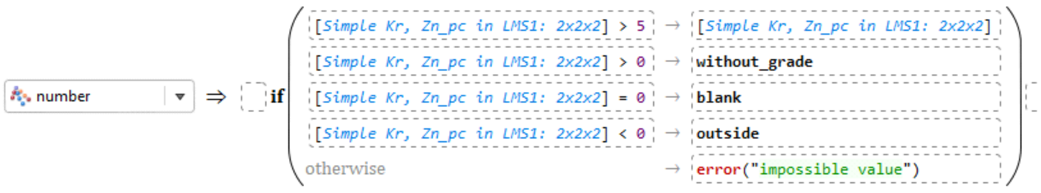
without_grade is often used to mean the estimator cannot produce a value

outside is used to indicate the block is outside the boundary of the domain

error generates an error, and provides an 'error' status value for the affected block

error('message') is similar to error but includes a custom message.

Multiple case example



```
number ⇒ if ( [Simple Kr, Zn_pc in LMS1: 2x2x2] > 5 → [Simple Kr, Zn_pc in LMS1: 2x2x2]
              [Simple Kr, Zn_pc in LMS1: 2x2x2] > 0 → without_grade
              [Simple Kr, Zn_pc in LMS1: 2x2x2] = 0 → blank
              [Simple Kr, Zn_pc in LMS1: 2x2x2] < 0 → outside
              otherwise                             → error("impossible value") )
```

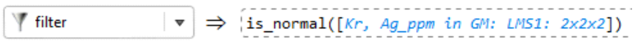
Explanation

if the Kriging estimate is greater than 5, the result will be the Kriging estimate. If the Kriging estimate is between 0 and 5, the block will be marked as having a without_grade result and status. If the Kriging estimate is exactly 0, the block will have a result and status of blank. If the Kriging estimate is negative, the block will have a result and status of outside. This should cover all the valid cases for the Kriging estimate value, but the if(::) If block requires an otherwise clause. In this case, the result of the otherwise will be to produce an error result and status with the message "impossible value".

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([Kr, Ag_ppm in GM: LMS1: 2x2x2])
```

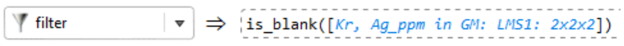
Explanation

If [Kr, Ag_ppm in GM: LMS1: 2x2x2] produces a normal value for a block, 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([Kr, Ag_ppm in GM: LMS1: 2x2x2])

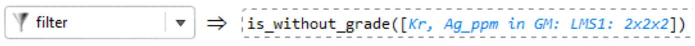
Explanation

If [Kr, Ag_ppm in GM: LMS1: 2x2x2] produces a blank status for a block, filter will be true for that block. If it produces any other invalid value or a normal value, filter will be false.

is_without_grade(*a*)

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

Example


filter ⇒ is_without_grade([Kr, Ag_ppm in GM: LMS1: 2x2x2])

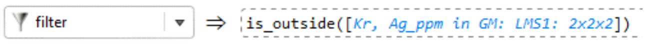
Explanation

If [Kr, Ag_ppm in GM: LMS1: 2x2x2] produces a without_grade status for a block, 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([Kr, Ag_ppm in GM: LMS1: 2x2x2])

Explanation

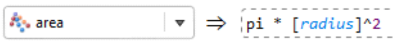
If [Kr, Ag_ppm in GM: LMS1: 2x2x2] produces an outside status for a block, filter will be true for that block. If it produces any other invalid value or a normal value, filter will be false.

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
```

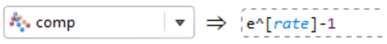
Explanation

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
```

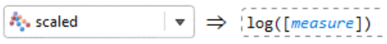
Explanation

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])
```

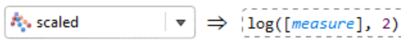
Explanation

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)

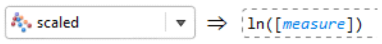
Explanation

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*])

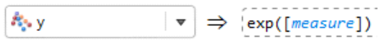
Explanation

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


y ⇒ exp([*measure*])

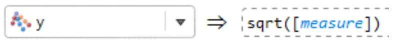
Explanation

y will be calculated as $e^{\text{[measure]}}$.

sqrt(*n*) ***Square root***

The principle square root of the provided number n.

Example


y ⇒ sqrt([*measure*])

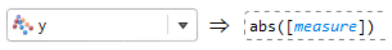
Explanation

y will 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



```
y ⇒ abs([measure])
```

Explanation

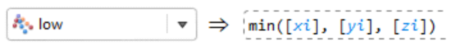
y will be whatever [measure] is, but without its sign; it will always be positive as a result.

Limits and Rounding

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

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

Example



```
low ⇒ min([xi], [yi], [zi])
```

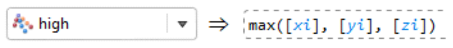
Explanation

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

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

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

Example



```
high ⇒ max([xi], [yi], [zi])
```

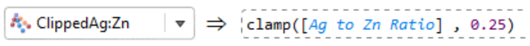
Explanation

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

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



```
ClippedAg:Zn ⇒ clamp([Ag to Zn Ratio] , 0.25)
```

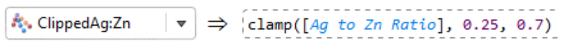
Explanation

The output for ClippedAg:Zn will range from 0.25 up to the maximum value of [Ag to Zn Ratio].If [Ag to Zn Ratio] is less than 0.25, the output will be 0.25 instead. Otherwise, the output will be [Ag to Zn Ratio].

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



```
ClippedAg:Zn ⇒ clamp([Ag to Zn Ratio], 0.25, 0.7)
```

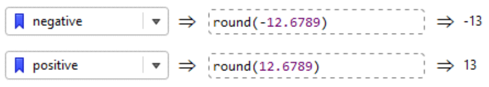
Explanation

The output for ClippedAg:Zn will range from 0.25 up to 0.7. If [Ag to Zn Ratio] is less than 0.25, the output will be 0.25 instead. If [Ag to Zn Ratio] is more than 0.7, the output will be 0.7 instead. Otherwise, the output will be [Ag to Zn Ratio].

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
```

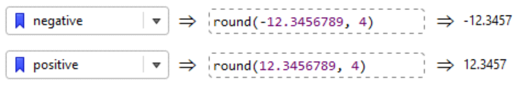
Explanation

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
```

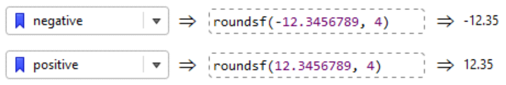
Explanation

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
```

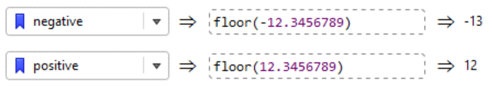
Explanation

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
```

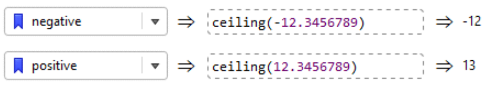
Explanation

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
```

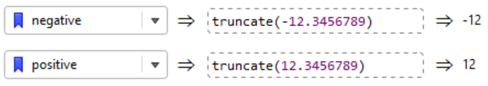
Explanation

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

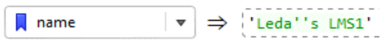
The variable negative will be given the value -12, the integer part of -12.3456789. The variable positive will be given the value 12, the integer part of 12.3456789.

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 Leda's LMS1 with an internal apostrophe, you might wrap the sequence with double quotation marks: "Leda's LMS1".

Example



name ⇒ 'Leda''s LMS1'

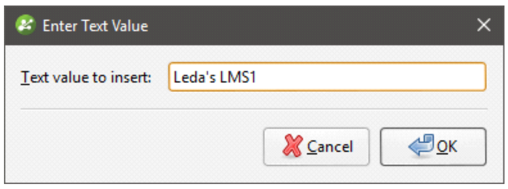
Explanation

The text sequence Leda's LMS1 (note: without the wrapping quotation marks and with only one possessive apostrophe) will be used wherever the variable name is used in expressions.

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

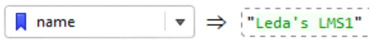


Enter Text Value

Text value to insert: Leda's LMS1

Explanation

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

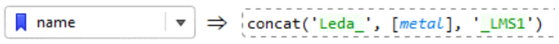


name ⇒ "Leda's LMS1"

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

This concatenates a series of text sequences together.

Example



name ⇒ concat('Leda_', [metal], '_LMS1')

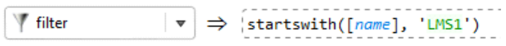
Explanation

Each of the text sequences in the input are run together and combined. If [metal] is 'Zinc' then name will be Leda_Zinc_LMS1. If [metal] is 'Silver' then name will be Leda_Silver_LMS1.

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], 'LMS1')
```

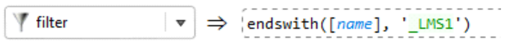
Explanation

filter will be true if [name] starts with LMS1.

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



```
filter ⇒ endswith([name], '_LMS1')
```

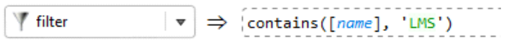
Explanation

filter will be true if [name] ends with _LMS1.

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



```
filter ⇒ contains([name], 'LMS')
```

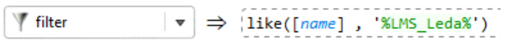
Explanation

filter will be true if [name] contains LMS 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



```
filter ⇒ like([name] , '%LMS_Leda%')
```

Explanation

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

- LMS1Leda
- LMS1LEDA
- LMS2Leda
- MoreLMS1Leda
- LMS3LedaExtra
- #lmsXleda#

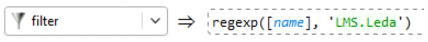
Examples that will not match include:

- LMS12Leda
- LM1SLeda
- LMSLeda

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], 'LMS.Leda')
```

Explanation

filter will be true if [name] matches the regexp pattern LMS.Leda. Examples of [name] that will match include:

- LMS1Leda
- LMS1LEDA
- LMS2Leda
- MoreLMS1Leda
- LMS3LedaExtra
- #lmsXleda#

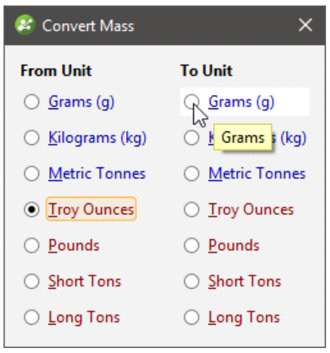
Examples that will not match include:

- LMS12Leda
- LM1SLeda
- LMSLeda

Unit Conversion

Mass/weight...

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

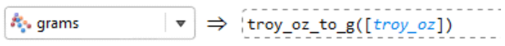


Convert Mass

From Unit	To Unit
○ Grams (g)	○ Grams (g)
○ Kilograms (kg)	○ Kilograms (kg)
○ 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



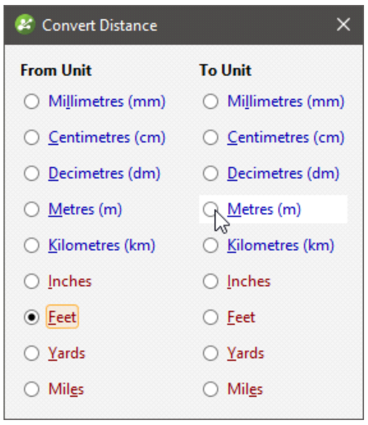
grams ⇒ `troy_oz_to_g([troy_oz])`

Explanation

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

Distance...

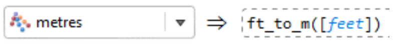
This selection opens a dialog box prompting for the **From Unit** and **To Unit** for the conversion.



From Unit: [✓ 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



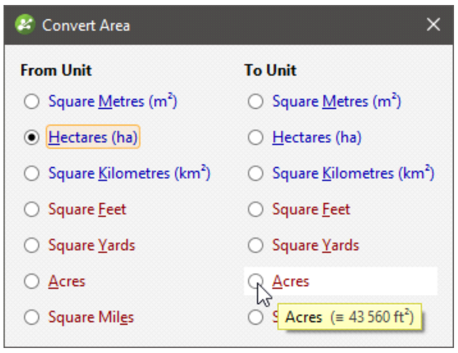
```
metres ⇒ ft_to_m([feet])
```

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 theft_to_m function.

Area...

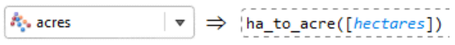
This selection opens a dialog box prompting for the **From Unit** and **To Unit** for the conversion.



From Unit: [✓ Hectares (ha)]

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

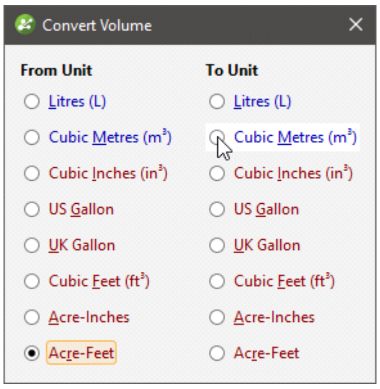

acres ⇒ ha_to_acre([hectares])

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 theha_to_ acre function.

Volume...

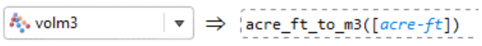
This selection opens a dialog box prompting for the **From Unit** and **To Unit** for the conversion.


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

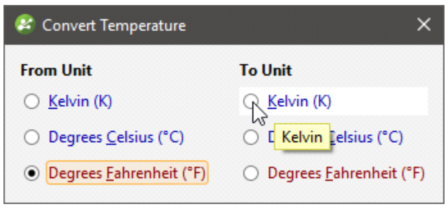

volm3 ⇒ acre_ft_to_m3([acre-ft])

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 theacre_ ft_to_m3 function.

Temperature...

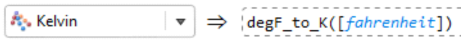
This selection opens a dialog box prompting for the **From Unit** and **To Unit** for the conversion.


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

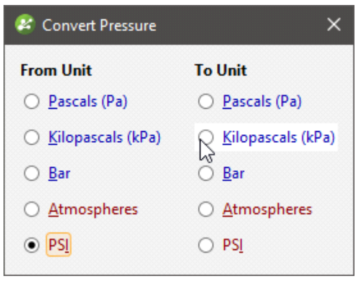

Kelvin ⇒ `degF_to_K([fahrenheit])`

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 thedegF_to_K function.

Pressure...

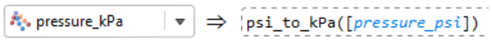
This selection opens a dialog box prompting for the **From Unit** and **To Unit** for the conversion.


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

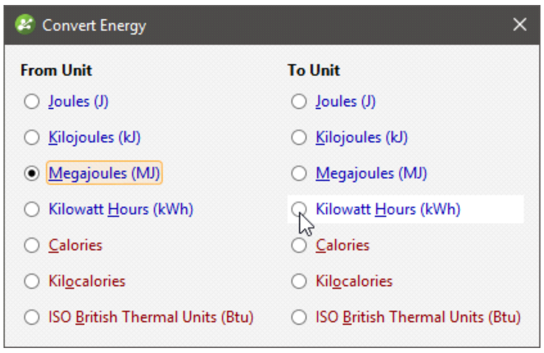

pressure_kPa ⇒ `psi_to_kPa([pressure_psi])`

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 thepsi_to_kPa function.

Energy...

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

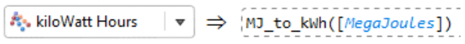


Convert Energy

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 theMJ_to_kWh function.