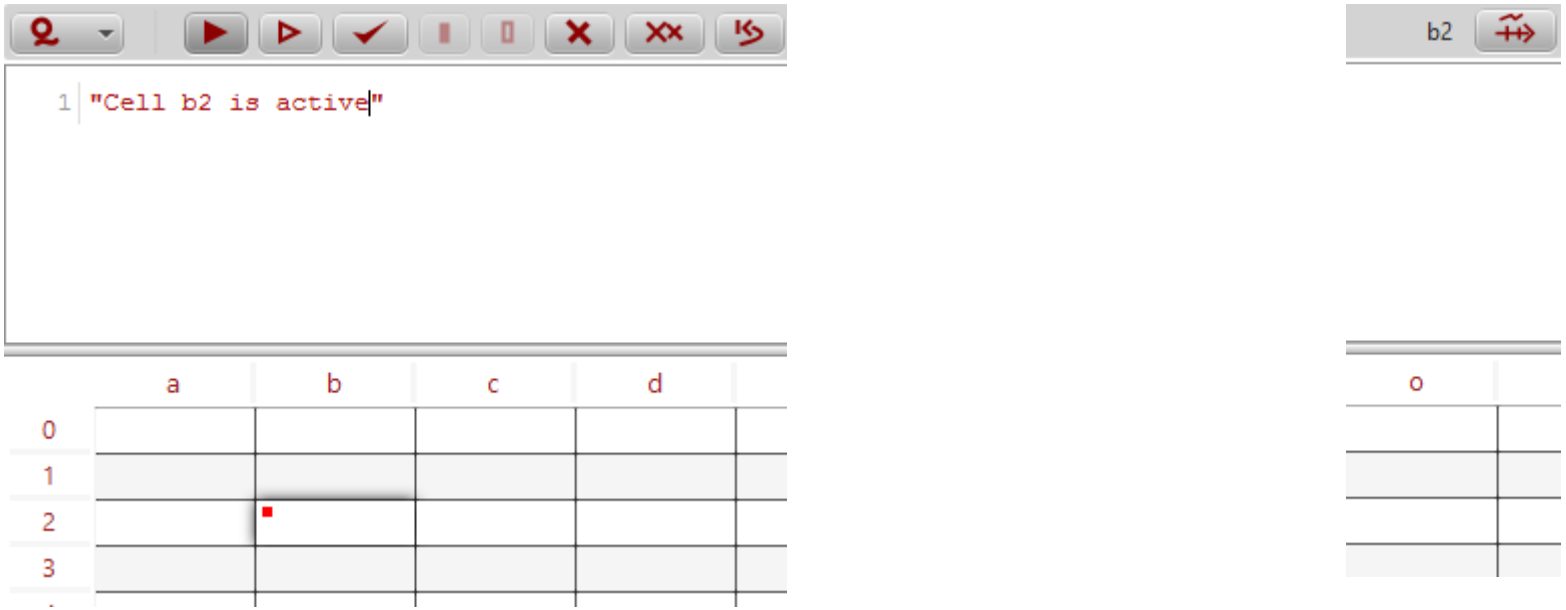
Cell Names

The location of the cell that is currently active is indicated by a red dot. The active cell name is also displayed in the top right corner of the spreadsheet.



Cell Color

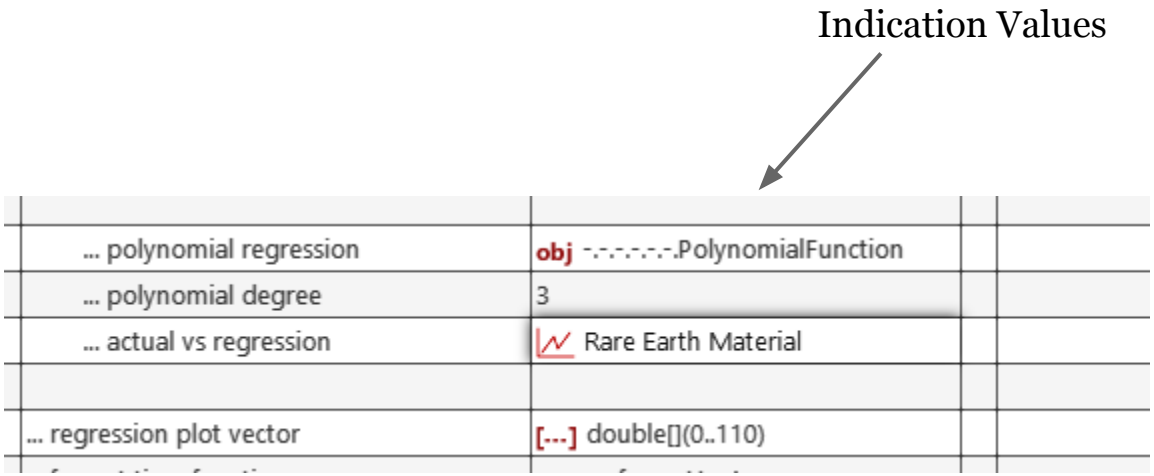
Cell color is used to indicate the status of calculations

Gray: Indicates that there is an error in the calculations or it has been stopped

Red to blue: Active cells are colored from red to blue with red cells currently being executed and blue cells waiting to be executed

Indication Value

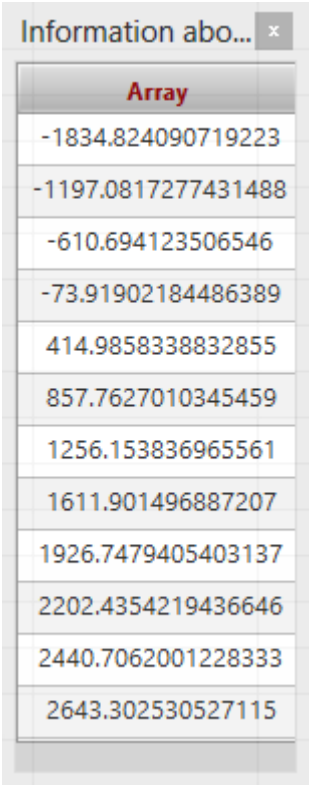
An indication value is used to display information about the type, value or state of the Java object in a given cell, providing the user with a visual representation of the cell object.



Full Value

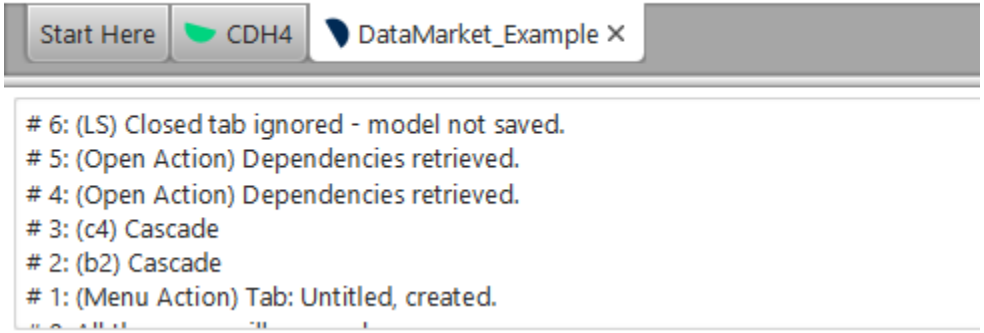
Further information about a cell can be obtained by double-clicking on it. This brings up the full value display for the cell object. The full value of a cell tries to communicate to the user the content of the object in a given cell as much as possible.

Example: A full value for an array object



Message Window

Messages, errors and general output from the environment is displayed in the message window. This information can be used to monitor and debug the current model being worked on. The message window is located at the bottom of the spreadsheet.



Shortcuts

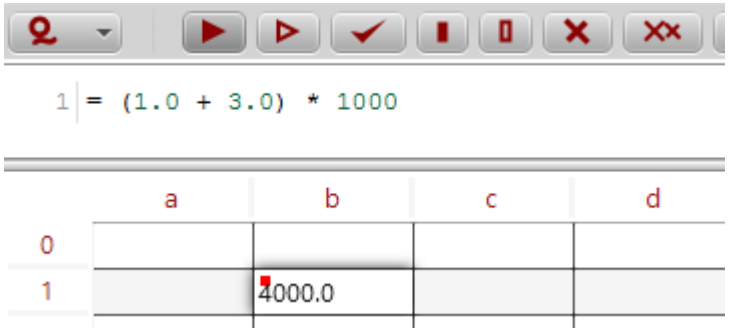
QuantCell allows the following shorthand notations for objects and primitive types. Examples of these shortcuts include ("=" is optional) ...

	... expression entered...	... result ...	... type ...
4	= 1	1.0	double
5	= 0.1	0.1	double
6	= true	01 true	boolean
7	= 1234I	1234	int
8	= 320F	320.0	float
9	= 835789237589347L	835789237589347	long
10	= 80%	0.8	double
11	=5B	5	byte
12	= 43S	43	short
13	="California"	California	String
14	='11/13/13 12:47 PM, GMT'	obj -.-.Date	Date
15	= 44D	44.0	double
16	= 0x123	291	long
17	= 0o123	83	long
18	='c'	c	char

Language Elements

It is just as easy to work with QuantCell as it is to work with any other spreadsheet. You write expressions, that return values, into cells and you combine these values in a meaningful way in your spreadsheet model.

You can work with Java syntax and benefit from the enormous Java ecosystem in QuantCell, but using QuantCell is not nearly as complicated as as writing Java code as a developer due to the user friendly spreadsheet approach employed.

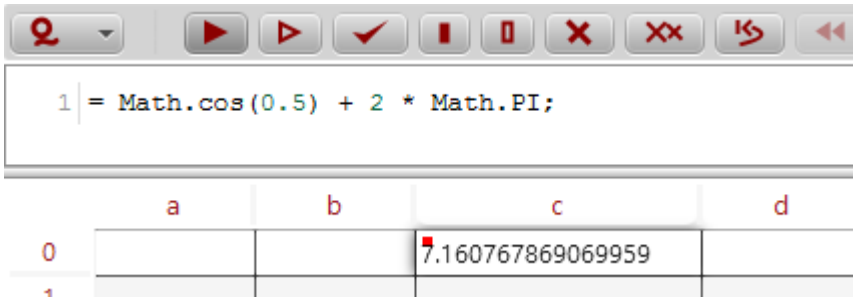


You can also use other powerful languages in QuantCell such as SQL for analysts and eventually both R and Scala to name a few.

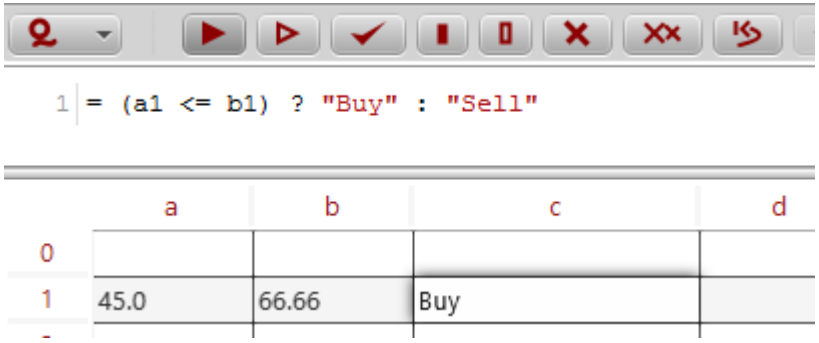
QuantCell has powerful methods to assist you in writing complicated formulas and in many cases will do the coding for you.

Simple Examples

Using Math functions



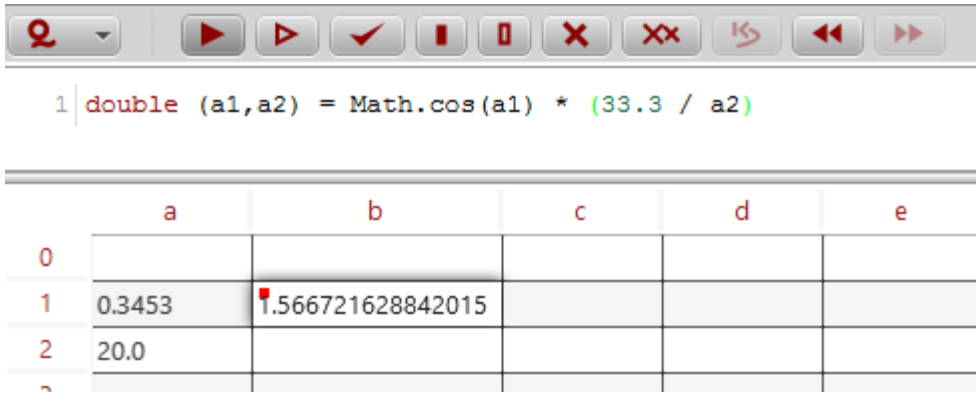
Using conditional expressions



Expressions

QuantCell is an expression oriented environment. The most common syntax for any formula is:

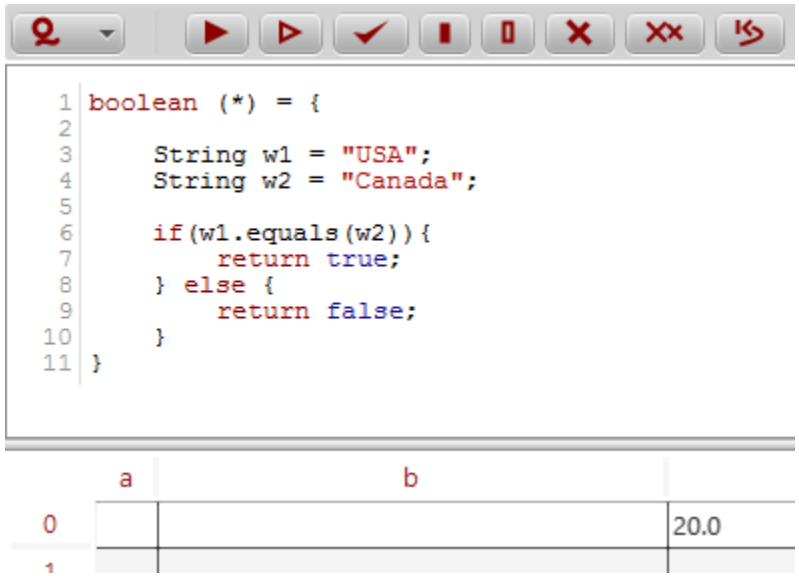
type (input parameters...) = expression;



If you don't want to worry about parameters, just use (*) or () for constant expressions. This tells the system to infer the parameters for you. The type is also inferred if not specified.

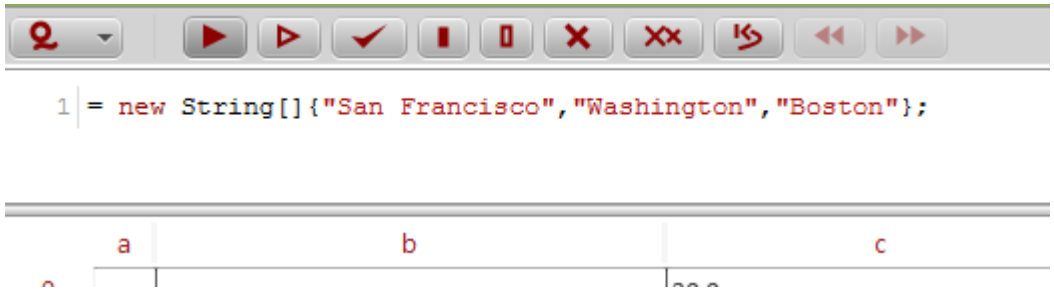
Using a Formula Block

It is often convenient to group statements together. This may be accomplished in QuantCell using a Java block that returns a value.



Types

It is not always necessary to specify the return type when writing a QuantCell formula. If the return type is not specified the system will try to infer it, defaulting to the Object type if unsuccessful. QuantCell is a statically typed system which eliminates many common spreadsheet errors.

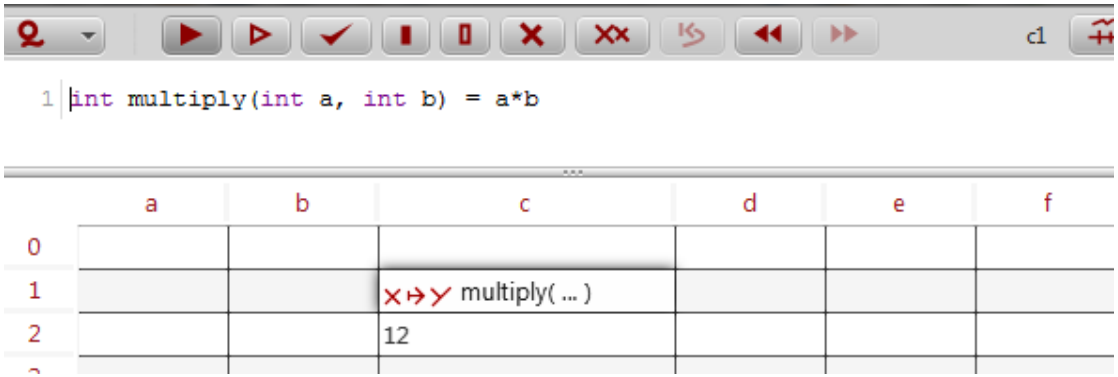


The default type for numbers in QuantCell is the Java primitive type "double".

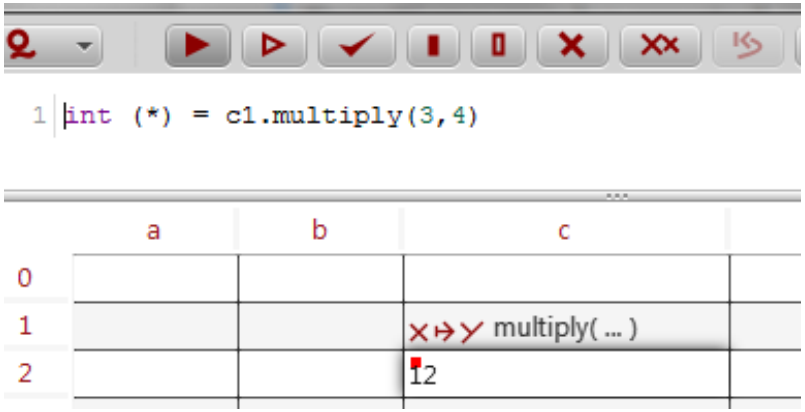
Functions

Functions can be created and used in QuantCell. The basic syntax for creating a function is:

type functionName (parameters…) = expression | return block

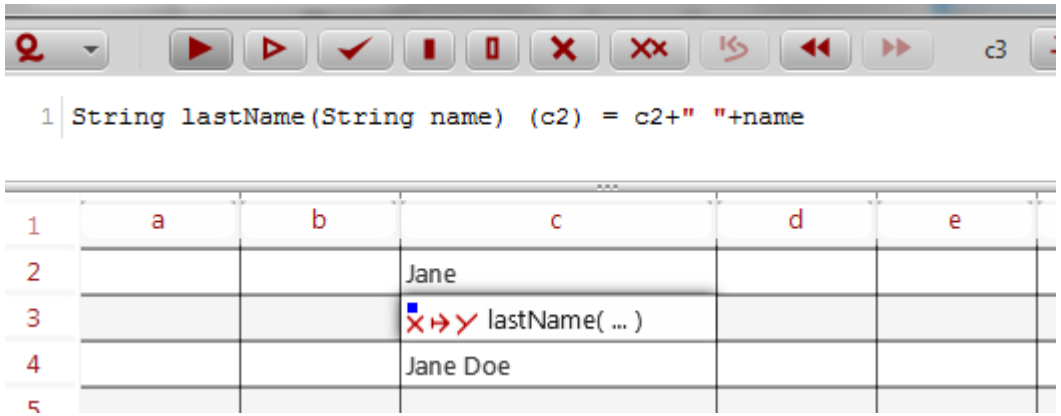


A user function may then be called from other cells by referencing the cell in which it was created …



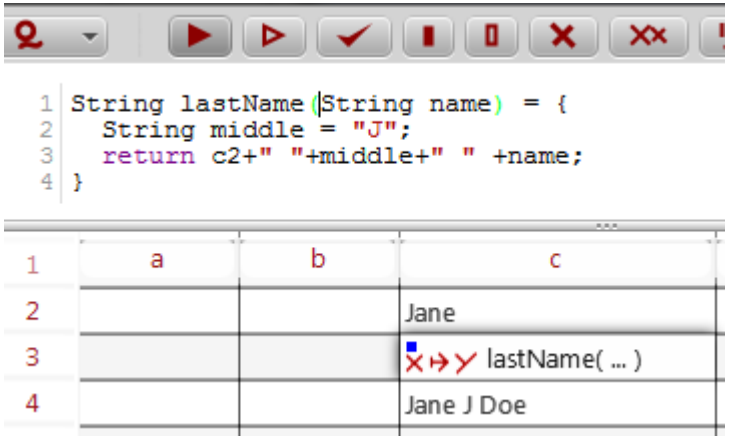
Functions (continue)

Functions can also reference values from other cells as well, as shown here to the right ...



for completeness, we should mention that this is a shorthand notation for the right hand side "String lastName(String name) (*) = ..." or the more explicit "String lastName(final String name) (c2) = ..." can also be used.

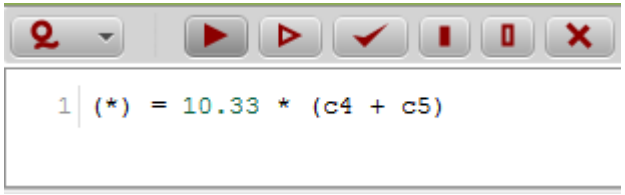
Functions may also be defined using return Java blocks, such as shown here ...



Variables

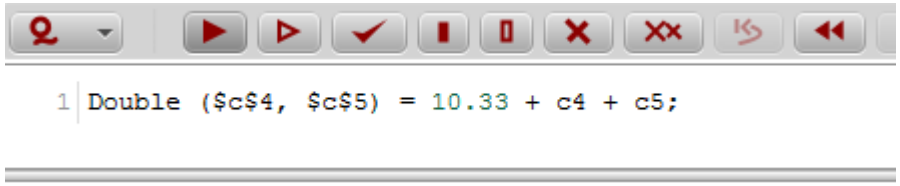
Objects and primitives in QuantCell are referenced using their respective cell name. If c4 and c5 hold an Integer value, the expression in another cell using these values can be:

To let the system infer the type and referenced parameters use:



Spreadsheet Notation

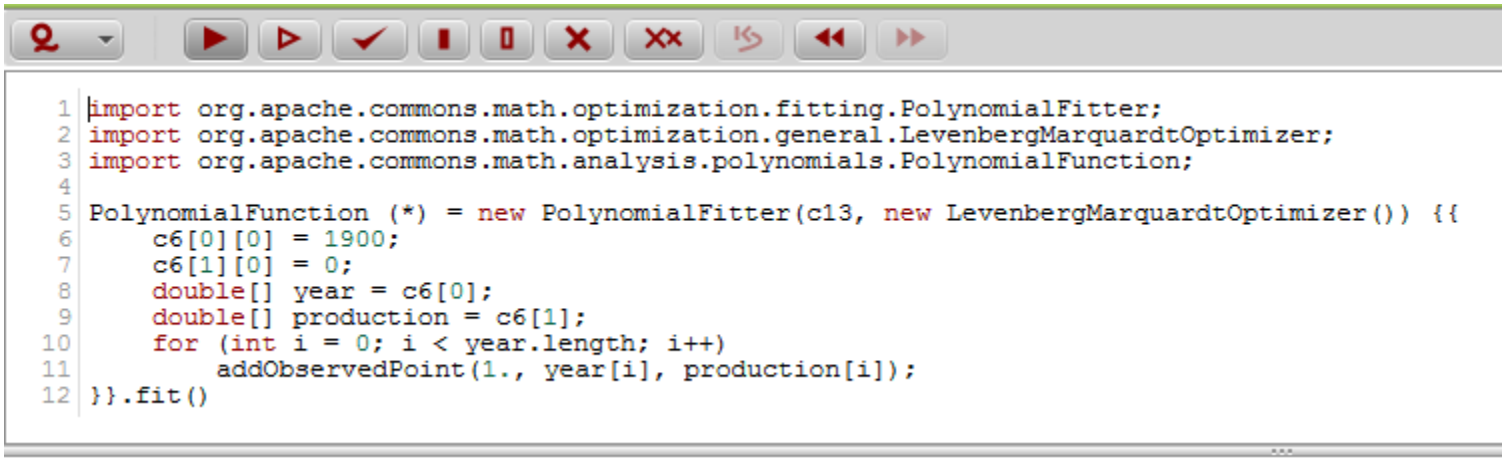
When referencing other cells as input parameters, a $ sign will cause the reference not to change when cells are copied or moved following a spreadsheet tradition.



Notice that although the spreadsheet "$" notation may be used for the input parameters, it is not used in the expression itself on the left hand side of the "=" symbol.

Imports

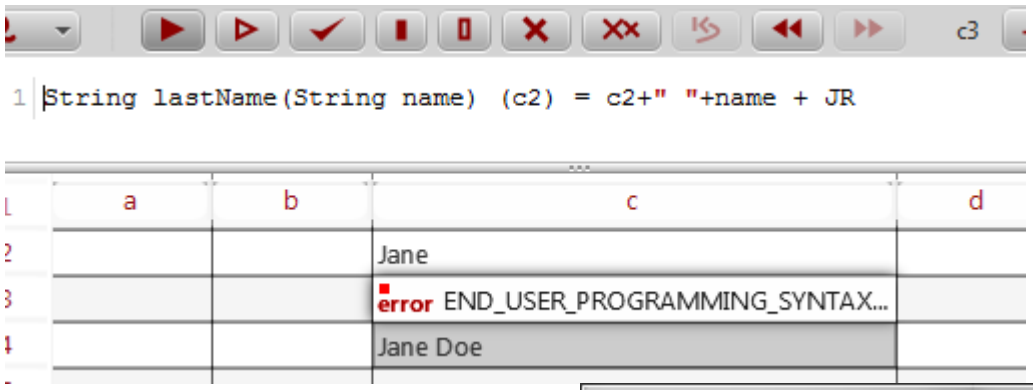
In order to use a Java library in a cell expressions, an import statement can be created to simplify references to the library. Instead of using an import statement the fully qualified name can also be entered directly in formulas. The imports are located first before the cell formula. Furthermore, the autocomplete functionality can be triggered to create these import statements automatically.



```
1  import org.apache.commons.math.optimization.fitting.PolynomialFitter;
2  import org.apache.commons.math.optimization.general.LevenbergMarquardtOptimizer;
3  import org.apache.commons.math.analysis.polynomials.PolynomialFunction;
4
5  PolynomialFunction (*) = new PolynomialFitter(c13, new LevenbergMarquardtOptimizer()) {{
6      c6[0][0] = 1900;
7      c6[1][0] = 0;
8      double[] year = c6[0];
9      double[] production = c6[1];
10     for (int i = 0; i < year.length; i++)
11         addObservedPoint(1., year[i], production[i]);
12 }}.fit()
```

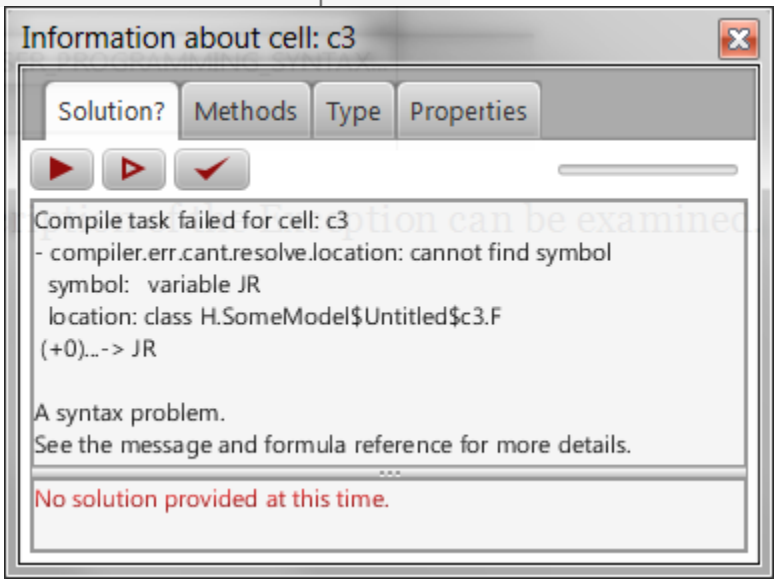
Errors and Exceptions

When a cell formula throws an exception it is indicated with the appropriate indication value ...



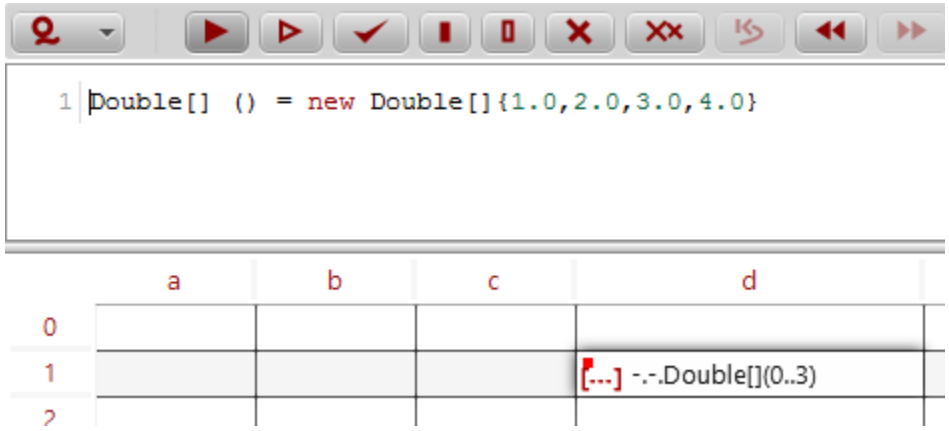
... by double-clicking on the cell a more detailed description of the exception can be examined, as shown here to the right ...

the spreadsheet will in many cases suggest a solution/fix to the problem.

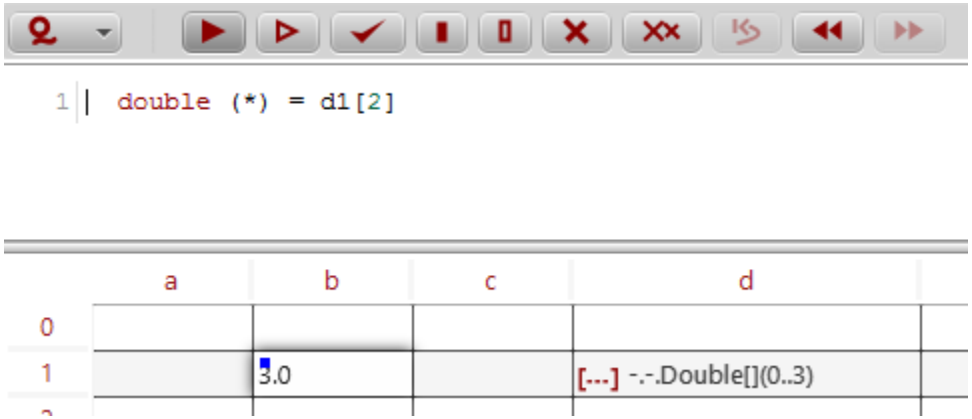


Arrays

Creating arrays in QuantCell is as simple as using any standard Java array notation, such as shown here ...



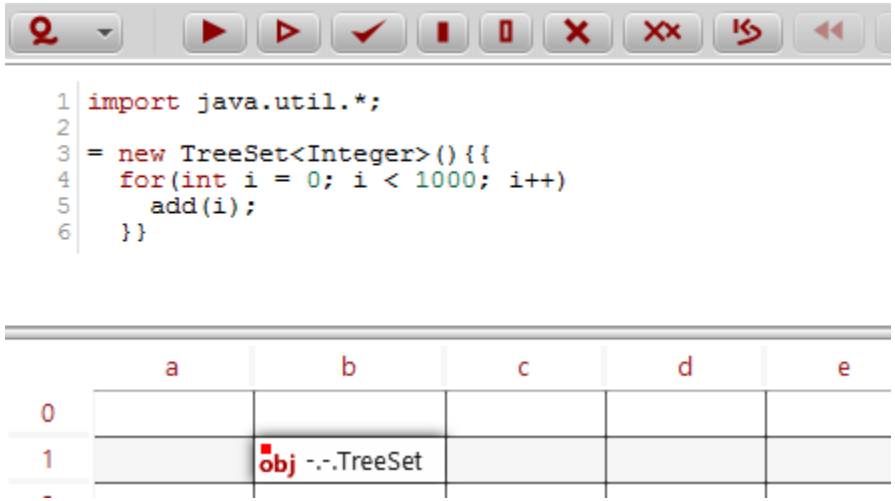
to reference the arrays or the array content, just use the standard notation again ...



Collections

Creating and using Sets, Lists, Maps and Queues in QuantCell follows standard Java notation ...

```
1 import java.util.*;
2
3 ArrayList () = new ArrayList<String>(Arrays.asList("Tesla","Ford","Toyota","Volvo"));
```

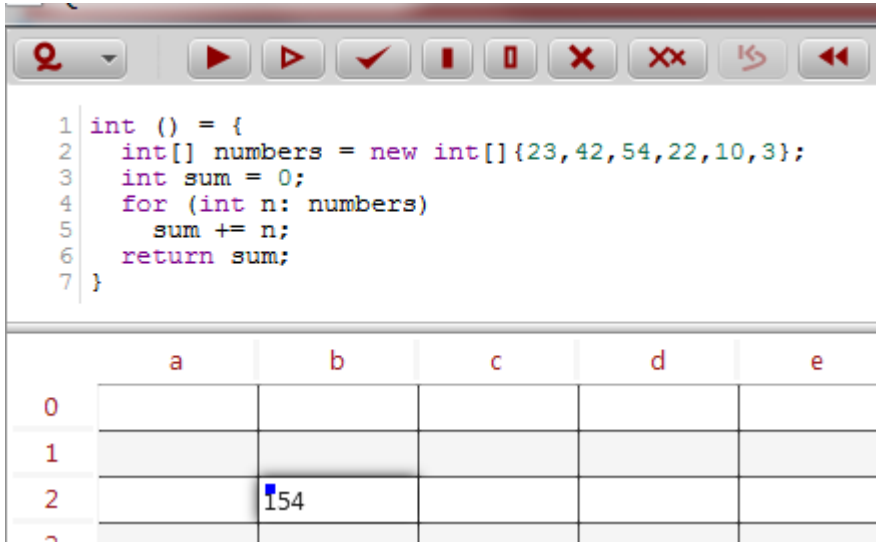


```
1 import java.util.*;
2
3 = new TreeSet<Integer>(){{
4   for(int i = 0; i < 1000; i++)
5     add(i);
6   }}
```

	a	b	c	d	e
0					
1		obj -.-.TreeSet			

Loops

Loops may be used in QuantCell ...



just follow the standard syntax from Java for "for","do","while" loops and so on.

Operators

Any operator allowed in the Java language can be used within a formula in QuantCell, here are commonly and not so commonly used examples ...

+, -, *, /, %, =, <, >, <=, >=, ==, !=, !, ? :, &, |, &&, ||

and then there are

+=, -=, *=, /=, ++, --

and eventually less commonly used operators such as

%=, &=, ^=, |=, <<=, >>=, >>>=, instanceof, ^, ~, <<, >>, >>>