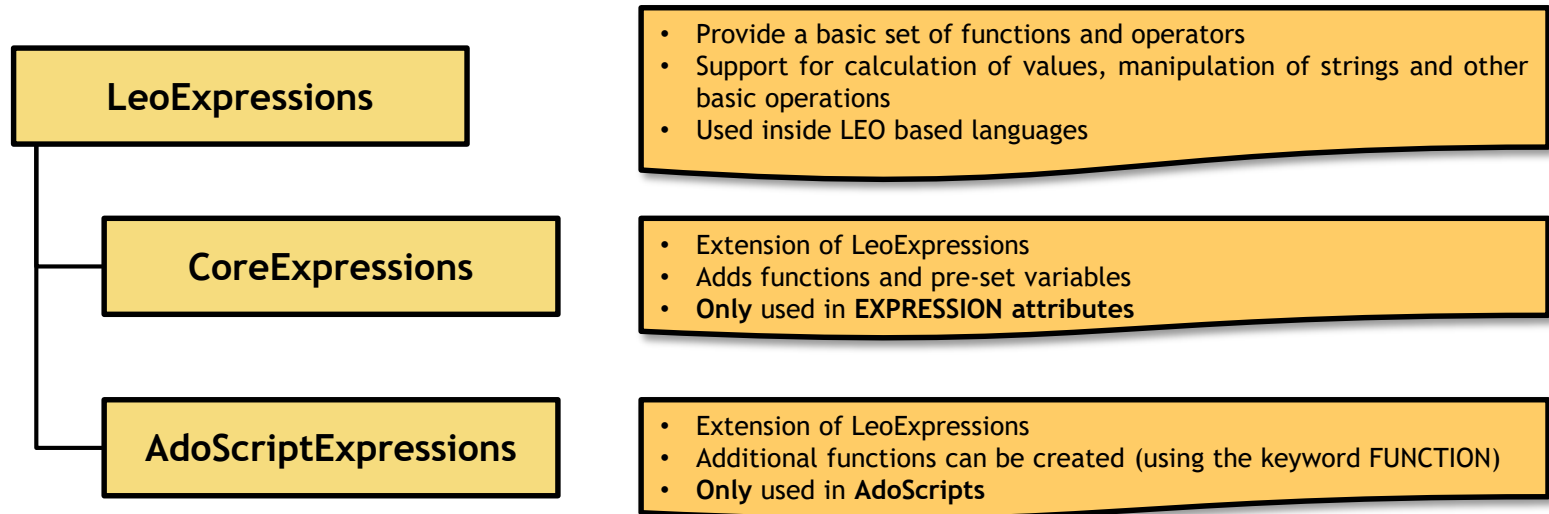


Expressions

ADOxx Expressions

- “Expressions” provide **a basic set of functions and operators** for investigating or calculating within a model context and across multiple models. This can be both a fixed calculation and the processing of attribute values.
- Exercises use the following library and models
 - "SmartCityLibrary-I-ModellingExercises.abl"
 - "SmartCity-I-SmartParking.adl"

Types of Expressions



- All three use:
 - "Simple Values" / Constants
 - Operators
 - Functions

Syntax of the language for EXPRESSION attributes

Syntax of the language for EXPRESSION attributes

ExprDefinition: `EXPR type: ResultType [format: FormatString]`
 `expr:[fixed:] CoreExpression`

ResultType: `double | integer | string | time`

fixed: the expression will be unchangeable in the Modelling Toolkit. It means that the EXPRESSION attribute evaluates the same formula everywhere.

Characteristics of Expressions

An expression attribute contains both a formula and the calculated value

There are two modes for using expression: fixed and editable

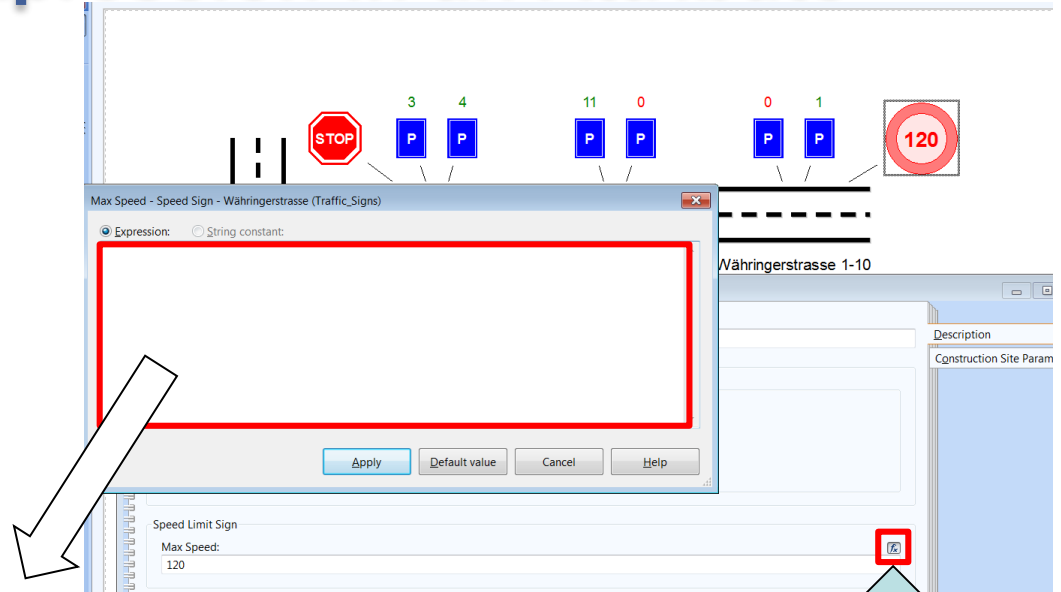
Fixed expressions store the formula in the default value of the attribute

An error message will be returned, if an error occurs when evaluating a formula

The last valid result is returned, if an inter-model expression can not be evaluated (when trying to access a not loaded model)

Expression attributes are always evaluated when an event occurs which can change the value. The changes are shown directly in the user interface

An Example of an Expression in an Attribute



random() returns
values between 0 and 1



cond (random () > 0.5 , 30 , 70)



If



Condition



Then



Else

Only available for
not fixed expressions

The last evaluated value is
taken as attribute value

An Example of an Expression in an Attribute in the Metamodel

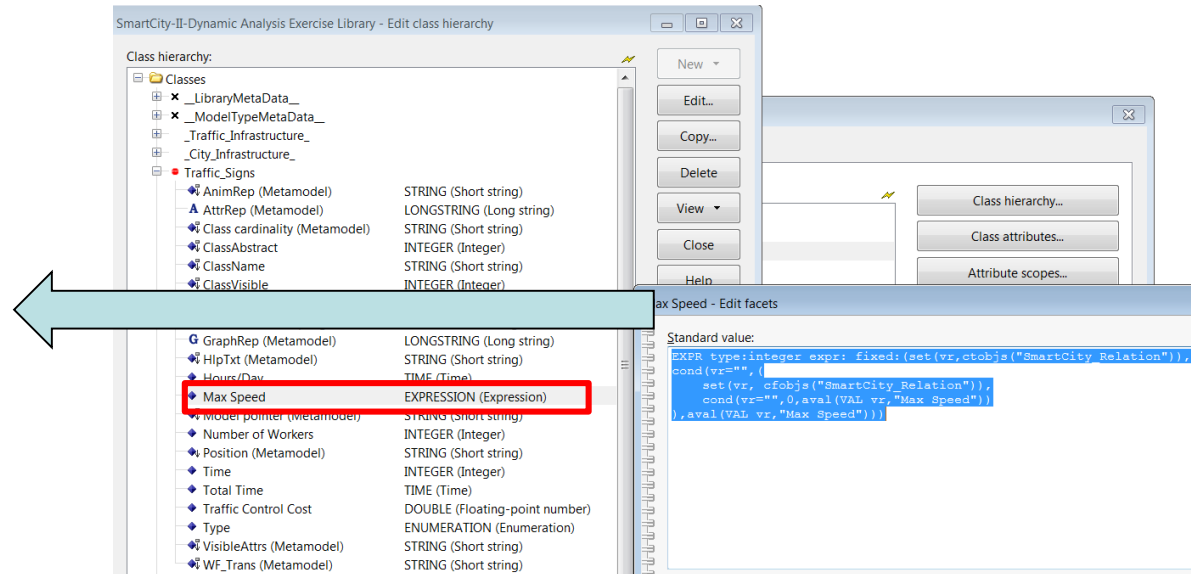
fixed Expression



EXPR type:integer
expr:fixed:(set(...



Fixed Expression Expression



An Example of an Expression in ADOscript

```
FUNCTION helloWorld world:string  
  return: (max(1,100))
```

 expression using max function

```
#call function
```

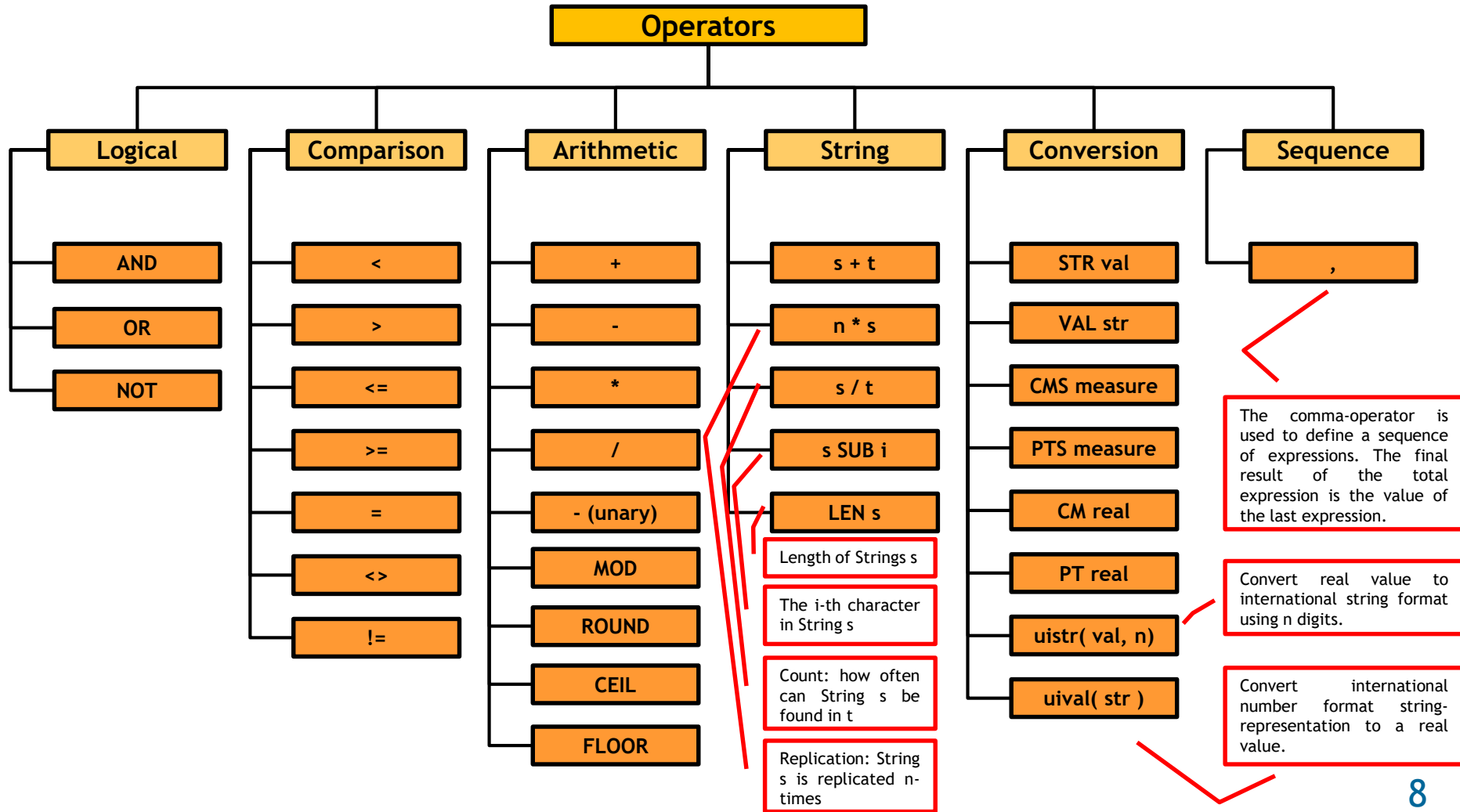
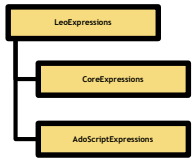
```
SET hello:(helloWorld("world"))
```

 expression calling a function

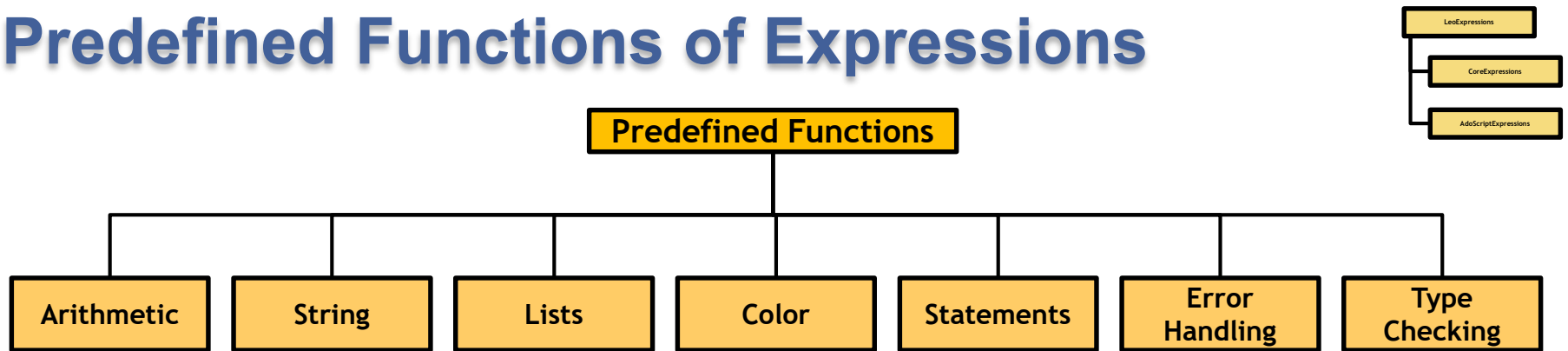
```
CC "AdoScript" INFOBOX (hello)
```

 expression evaluating variable

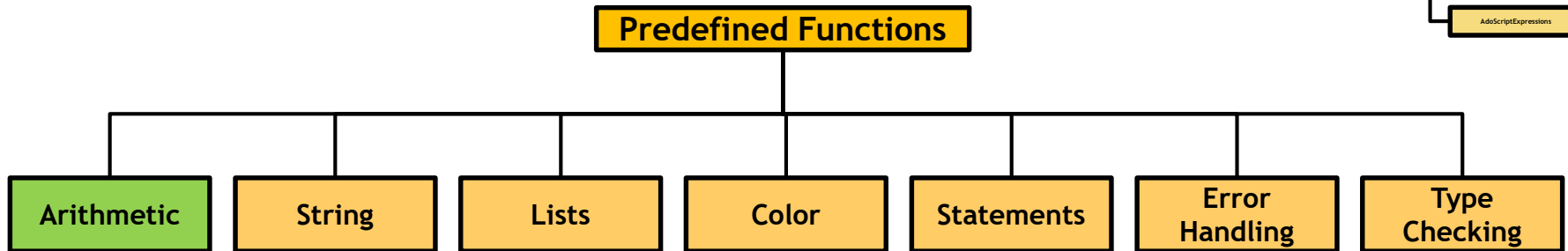
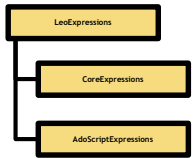
Operators of Expressions



Predefined Functions of Expressions

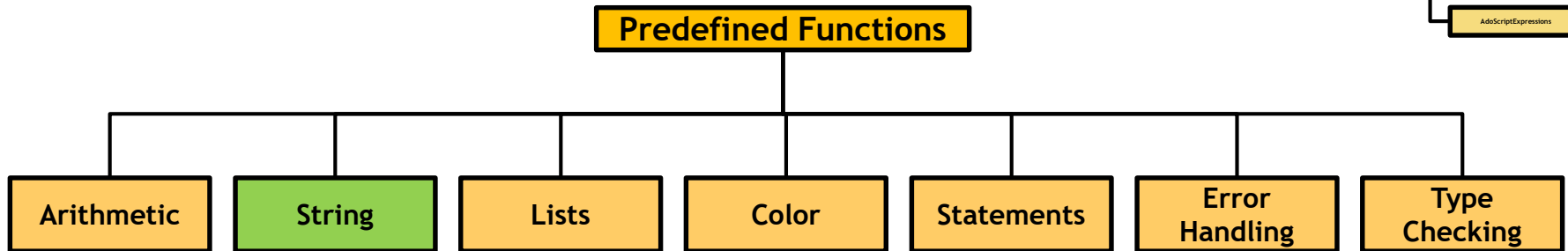
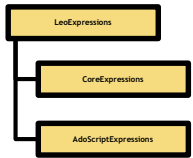


Predefined Functions of Expressions



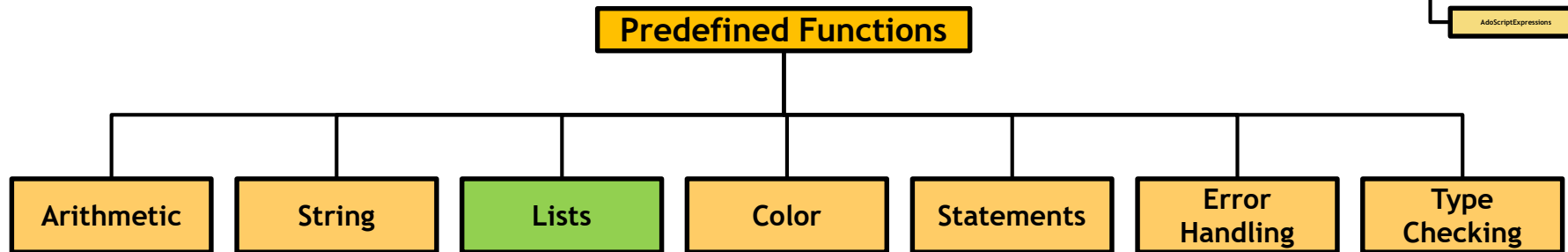
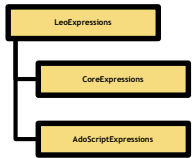
<code>abs(x)</code> <code>max(x,y) min(x,y)</code> <code>pow(x,y) sqrt(x)</code> <code>exp(x)</code> <code>log(x) log10(x)</code>	Arithmetic functions
<code>sin(x) cos(x) tan(x)</code> <code>asin(x) acos(x) atan(x)</code> <code>sinh(x) cosh(x) tanh(x)</code>	Trigonometric functions
<code>random()</code>	Random value $0 \leq n < 1$
<code>round(x)</code>	Round-to-nearest, i.e. if decimal ≥ 0.5
<code>floor(x) ceil(x)</code>	Round up/down

Predefined Functions of Expressions



<code>search(source, pattern, start)</code>	Searches in <i>source</i> for <i>pattern</i> , starting at <i>start</i> (0-based), returns index or -1
<code>bsearch(source, pattern, start)</code>	Search begins at end of source string (backwards)
<code>copy(source, from, count)</code>	Copies <i>count</i> characters from <i>source</i> beginning at <i>from</i> (0-based)
<code>replall(source, pattern, new)</code>	Replaces all occurrences of <i>pattern</i> in <i>source</i> with <i>new</i>
<code>lower(source)</code>	Transforms to lower-case
<code>upper(source)</code>	Transforms to upper-case
<code>mstr(string)</code>	Puts the string between “” and escapes special characters

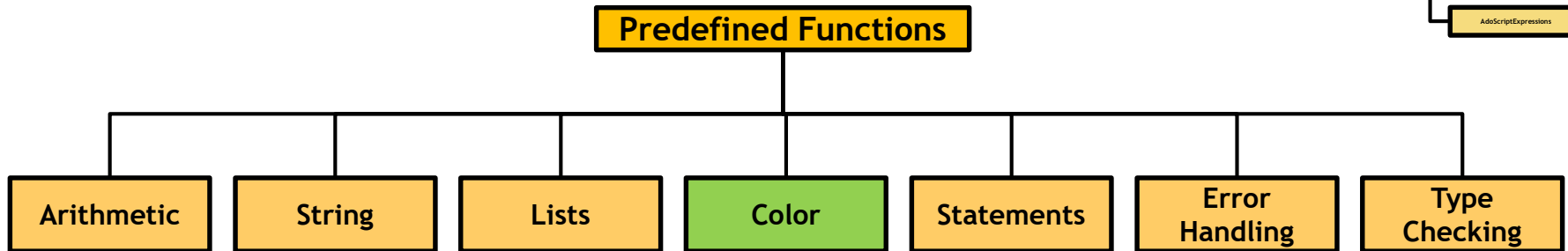
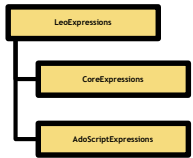
Predefined Functions of Expressions



<code>token (source , index [, separator])</code>	token at specific index in source separated by separator (default = single whitespace)
<code>tokcnt (source [, seperator])</code>	Counts tokens in <i>source</i> separated by <i>seperator</i> (default = single whitespace)
<code>tokcat (source1 , source2 [, separator])</code>	Concatenates two lists
<code>tokunion (source1 , source2 [, separator])</code>	Union of two lists
<code>tokisect (source1 , source2 [, separator])</code>	Intersection of two lists
<code>tokdiff (source1 , source2 [, separator])</code>	Difference of two lists

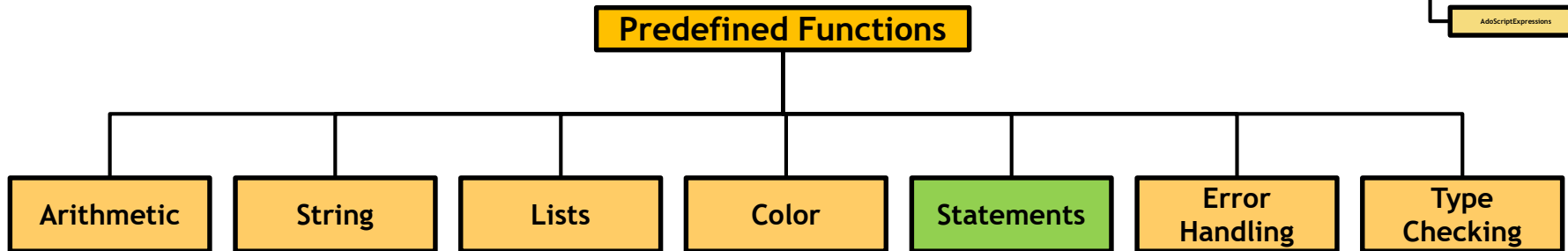
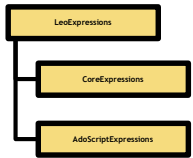
A list is a string of tokens separated typically with a single space, e.g. a list of object IDs:
"13452 13574 13845"

Predefined Functions of Expressions



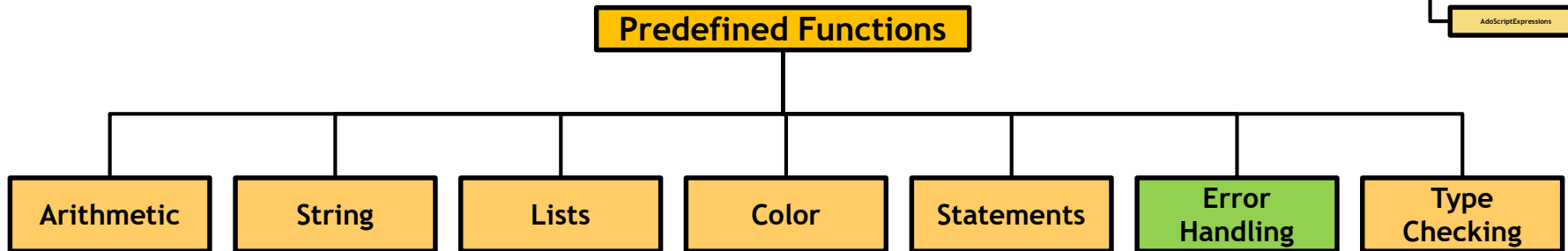
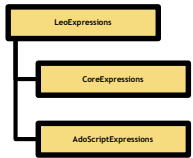
<code>rgbval(colorname)</code>	24bit RGB-Value of the color (by name)
<code>rgbval(r,g,b)</code>	Calculates the RGB-Value for the provided color values.

Predefined Functions of Expressions



<code>set (var, expr)</code>	<i>Expr</i> will be stored in <i>var</i> . Variable <i>var</i> is created implicitly.
<code>cond (cond1, expr1, ..., expr_else)</code>	Evaluate <i>cond1</i> , if true return <i>expr1</i> , if false return next condition or return <i>expr_else</i> .
<code>while (cond, loopexpr [, resultexpr])</code>	While <i>cond</i> is true, evaluate <i>loopexpr</i> . Return <i>resultexpr</i> .
<code>fortok (varname, source, sep, loopexpr [, resultexpr])</code>	For each element in the list <i>source</i> , evaluate <i>loopexpr</i> . The current element is stored in <i>varname</i> . The list elements are separated by <i>sep</i> . Return <i>resultexpr</i> .
<code>for (lvalue, from, to, loopexpr [, resultexpr])</code>	For in numeric form, <i>from</i> a number <i>to</i> a number.

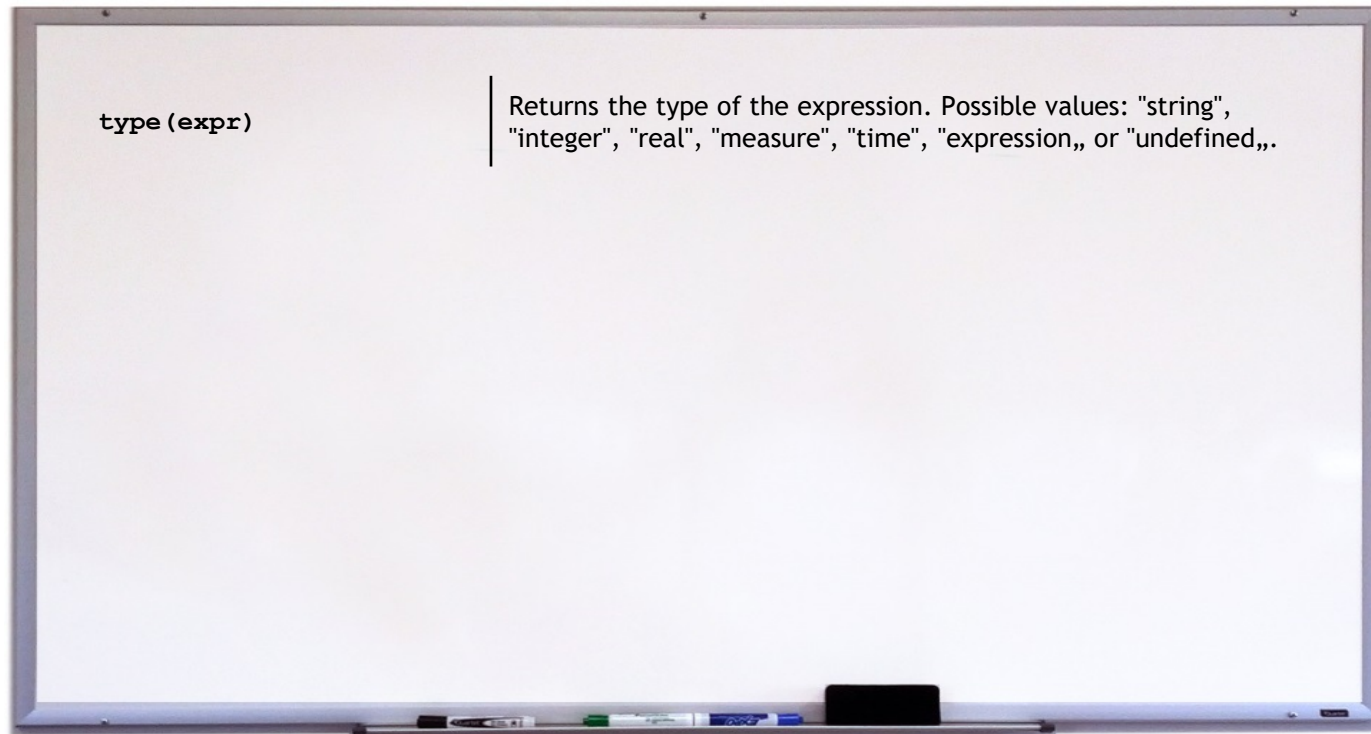
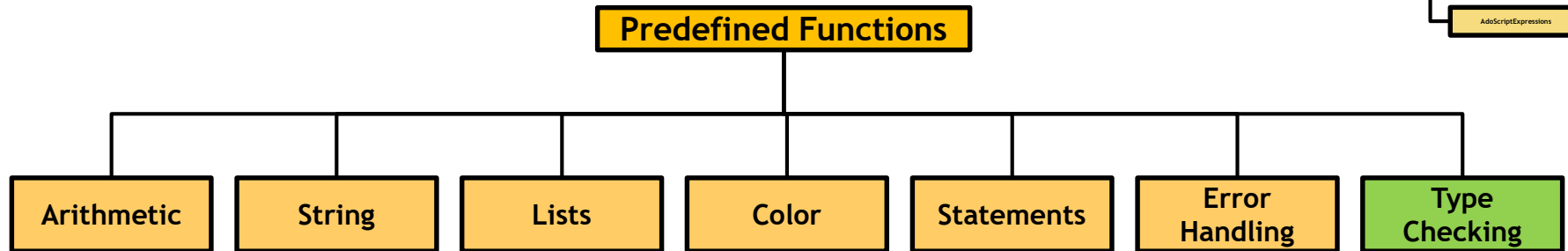
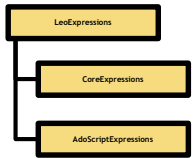
Predefined Functions of Expressions



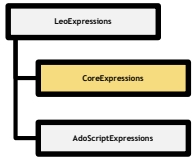
`try(expr, failexpr)`

Returns *expr*, if it succeeds, otherwise returns *failexpr*.

Predefined Functions of Expressions

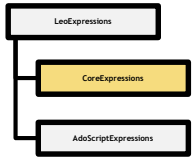


Selected Core Expressions functions



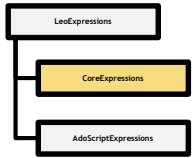
<code>aval (instID, attrName)</code> <code>aval (attrName)</code> <code>aval (instName, attrName)</code> <code>aval (className, instName, attrName)</code>	Call the value of the instance attribute.
<code>asum (idStr, attrName)</code>	Returns the sum for all objects specified by idStr of all attribute values of the attribute specified by attrName.
<code>irtmodels(attrName)</code> <code>irtmodels(instID, attrName)</code> <code>irtmodels(instName, attrName)</code> <code>irtobjs(attrName)</code> <code>irtobjs(instID, attrName)</code> <code>irtobjs(instName, attrName)</code>	Returns the models/objects referenced from the specified instance INTERREF attribute attrName.
<code>aql(aqlExpr)</code>	Returns a string with the IDs of the AQL expression's result.

Selected Core Expressions functions



<code>class(instID)</code> <code>class(className)</code>	Returns the class ID
<code>allobjs(modelID, className)</code>	Returns all objects of the class <code>className</code> in the specified model <code>modelID</code> .
<code>conn(relnName, fromObjs, toObjs)</code>	<p>Returns the connectors of the specified type <code>relnName</code> between the specified objects.</p> <p>An <code>objs</code> parameter can be a single ID or a string containing a sequence of IDs.</p>
<div> <div> <code>ctobj()</code> <code>cfobj()</code> <code>ctobjs(relnName)</code> <code>ctobjs(instid,relnName)</code> <code>cfobjs(relnName)</code> <code>cfobjs(instid,relnName)</code> </div> <div> <div>of relations</div> <div>of objects</div> </div> </div>	Returns the to/from-object

Selected Core Expressions functions

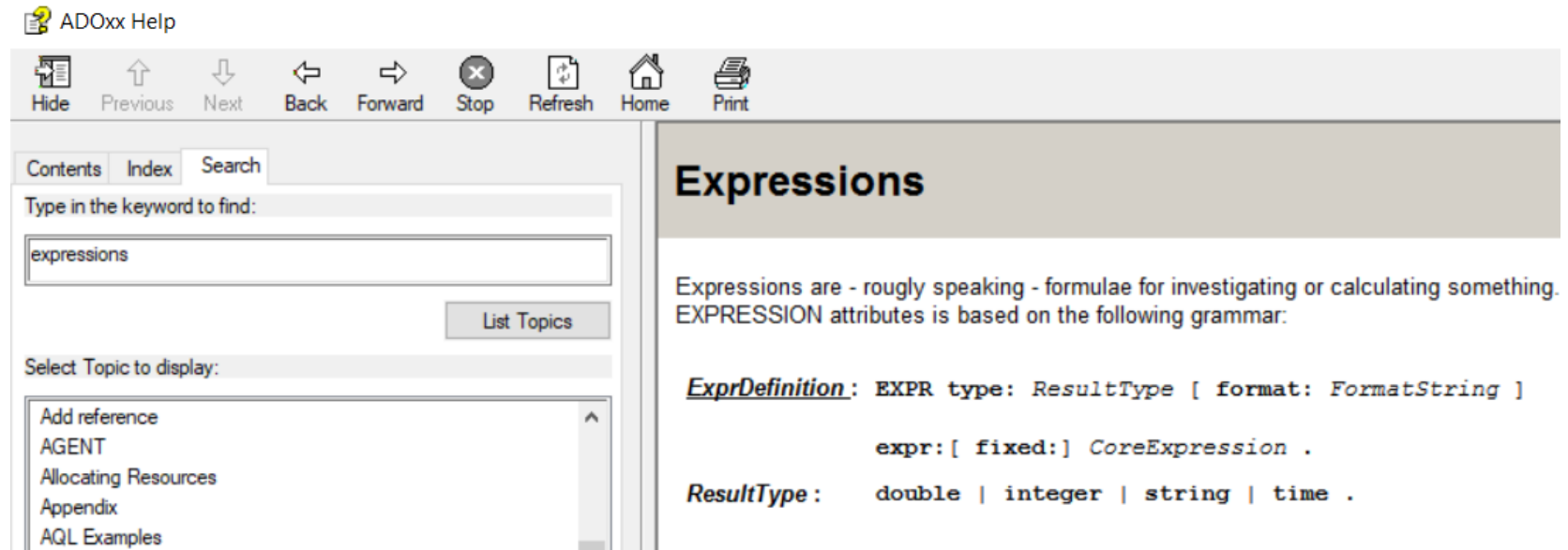


Additionally, the following variables are preset for every **EXPRESSION** attribute:

- *objid*: containing the ID of the own object/connector
- *classid*: containing the ID of the own class/relation class
- *modelid*: containing the ID of the model containing the own object
- *attrid*: containing the ID of this attribute

More functions

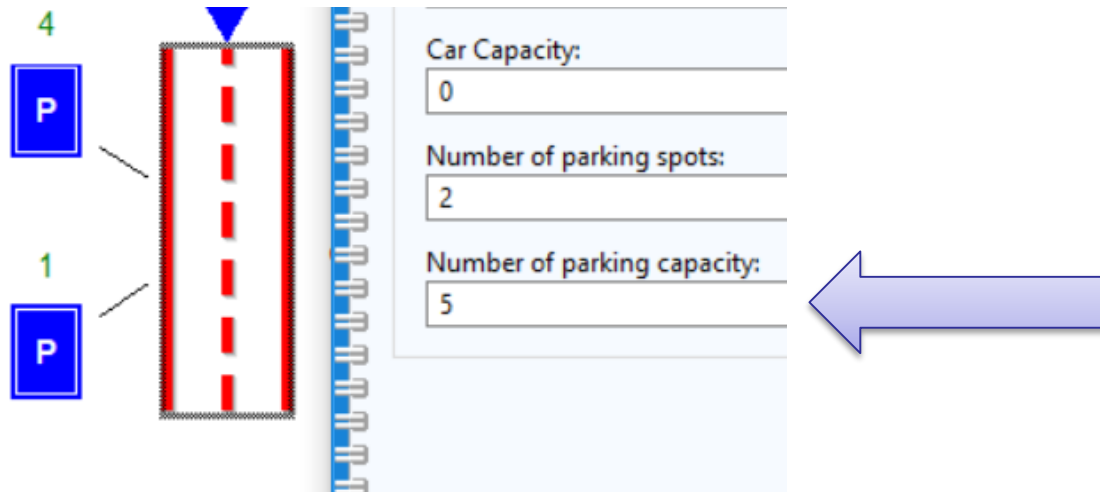
- You can find all functions in the ADOxx help and on <https://www.adoxx.org/live/expressions>



The screenshot shows the ADOxx Help web interface. At the top, there is a navigation bar with icons for Hide, Previous, Next, Back, Forward, Stop, Refresh, Home, and Print. Below this, there are tabs for Contents, Index, and Search. The Search tab is active, and a search box contains the text 'expressions'. A 'List Topics' button is located below the search box. To the left of the main content area, there is a 'Select Topic to display:' section with a list of topics: Add reference, AGENT, Allocating Resources, Appendix, and AQL Examples. The main content area is titled 'Expressions' and contains the following text: 'Expressions are - roughly speaking - formulae for investigating or calculating something. EXPRESSION attributes is based on the following grammar:'. Below this, there is a grammar definition:
ExprDefinition: `EXPR type: ResultType [format: FormatString]`
`expr:[fixed:] CoreExpression .`
ResultType: `double | integer | string | time .`

Exercise – Smart Parking

- Use the SmartCity-I Library available on Moodle
 - Create a new expression attribute in class ,Road‘, which calculates the total number parking capacity of all connected parking spots.



**THANK YOU FOR YOUR
ATTENTION**

