List Functions

There also exist algebraic operators (+, -) to concatenate or subtract lists.

asList(Object o)

Makes a cast to the object o to state, that this object is a list. Cannot be used for list creation. Use toList for this.

addTo(Object o, List l)

Adds an element to an existing list.

+ (append a list, "allAll")

Usage: <list1> + <list2>

Adds the second list at the end of the first list. Example:

```
#PARSER PQL
#RUNQUERY

input = TIMER({period = 1000,timefromstart = true,source = 'Source'})

lists1 = MAP({expressions = [['addTo(42, toList(1,2,3))','list1']]},input)

lists2 = MAP({expressions = [['list1 + toList(10,20,30)','list2']],
keepinput = true}, lists1)
```

toList(Object o)

Creates a new list with given input o.

Remark: You can also use ... with up to 20 arguments:

toList(Object o, Object o),toList(Object o, Object o, Object o), ..., toList(Object o, Object o, Object o, Objec t o, Object o, Object o, Object o, Object o, Object o, Object o)

split(String list, String delimiter)

Creates from the csv-input string a new list by using the delimiter. The type of the list elements will be Object.

split(String list, String delimiter, String datatype)

Creates from the csv-input string a new list by using the delimiter. The type of the list elements will be datatype.

fill(List list, Number desiredListSize, Object listDefaultElement)

Checks if the list `list` has the desired size. If not, the list will be enlarged. `listDefaultElement` is used as list element for the created spots in the list.

<List>[Number x]

Returns the object on position *x* of the given list, e.g. 'timestamps[0]'.

IndexOf(List l, Object o)

Returns the index of the given object in the given list.

IsEmpty(List l)

Return true if the given list is empty else false.

Size(List l)

Return the size of the list.

isEmpty(List l)

returns true, if the list contains no values.

elementAt(List l, int pos)

Return the element of the list l at pos. If the list is a variable or a constant list[pos] can be used.

first(List l)

returns the first element of the list. Same as <List>[0] or elementAt(l,0)

last(List l)

returns the last element of the list.

rest(List l)

returns a new list containing every element, except the first element.

sublist(List l, Number from [, Number to])

Returns a new List containing the elements from position from to position to. If to is not given, the function returns the sublist, starting an from.

rnd(List l)

Returns a random element from the list.

contains(List l, Object elem)

Returns true, if element is part of the list. Equivalent to indexOf(l, elem) >= 0.

indexOf(List l, Object elem)

Returns the position of the object elem inside the list. Returns -1 if elem is not found

search(List l, Object o, Boolean useNextOne)

Search for an element o in a sorted list. The elements must be comparable, else an exception is thrown.

It the value is not found and

- useNextOne is true, the next higher value (if available) will be used.
- useNextOne is false, the next lower value (if available) will be used.

For Input: 1,5,7,8,33,93 the following

```
map1 = MAP({
            expressions = [
                'search(list,92,false)',
                'search(list,93,false)',
                'search(list,92,true)',
                'search(list,93,true)'
            ]
          },
          list
        )
```

return 33,93,93,93

max(List l)

Returns the highest element in the list

max(List<Tuple> l, Number pos)

Returns the highest element in the list. Here the elements are treated as Tuples and pos is the number of the attribute that should be used for comparing

min(list l)

returns the lowest element in the list

min(List<Tuple> l, Number pos)

Returns the lowest element in the list. Here the elements are treated as Tuples and pos is the number of the attribute that should be used for comparing.

sum/avg(List l)

Returns the sum/avg of all values in the List.

filter(List l, String predicate)

Executes the predicate on each element in the list and returns a filtered list. The variable can be defined free.

```
///For list (1,5,7,8,33,93)
map2 = MAP({
            expressions = [
                'filter(list,"x>=8")'
            ]
          },
          list
        )
/// will return a new List with (8,33,93)
```

removeDuplicates(List l)

returns a new list where duplicates are removed

ListProject(List<Tuple> l, String positions)

Restricts each Tuple in the list to the given attribute positions. positions must be a comma seperated string (e.g., "1,4,7").

foreach(List l, String expression)

Perform the given expression on each element in the list and returns a list with the mapped elements.

foreachpair(List l, String expression)

Performs the given expression on each pair of input elements in the list, i.e. input list is treated as (e11, e12,e21,e22,e31,e32, ...)

When using list of tuples as input, this is a little more complex. Here is an example where all the prices are added pair wise (price is the 5th entry in the tuple)

```
#PARSER PQL
#ADDQUERY
bidsWindow = TIMEWINDOW({
                  size=[80, 'MILLISECONDS'],
                  advance=[80, 'MILLISECONDS']
                },
                bid
              )

reduce = AGGREGATION({
              aggregations = [['FUNCTION' = 'Nest']],
              eval_at_new_element = false,
              eval_before_remove_outdating = true
            },
            bidsWindow
          )
pairs = MAP({
            expressions = [
              ['foreachpair(nest, "elementAt(asTuple(a),4) + elementAt
(asTuple(b),4)")','pairs'], 'nest'
            ]
          },
          reduce
        )
```

all(List l, String expression)

Returns true, if the expression holds for all elements in the list

any(List l, String expression)

Returns true, if the expression is true for any element in the list