FEEL - Lists

What I Will Learn

What Will I Learn?

At the end of this course you will be able to:

- Understand the basic list data type
- Operate with various list expressions:
    - **Get**
    - **Filter**
    - **Some**
    - **Every**
- Use various number list functions:
    - min() and max()
    - sum() and product()
    - mean(), median(), and stddev()
    - mode()
- Use various general list functions:
    - list contains()
    - count()
    - all() and any()
    - sublist()
    - append(), insert before(), and remove()
    - reverse() and index of()
    - union(), distinct values(), and duplicate values()
    - sort()
    - flatten(), concatenate(), and string join()

List

Data Type

In the next lessons, we will cover the basics of the **list** data type in 🔗 FEEL used in Camunda, followed by an exploration of key list expressions and functions All these topics will be cover along the course.

Basic Concepts

A list in FEEL is an ordered collection of elements that can be of **any data type**. Lists are essential for performing operations on groups of elements and are fundamental to decision modeling.

Examples

Creating a List

To create a list in FEEL, you enclose the elements within square brackets [].

// empty list

[]

// number list

[1, 2, 3, 4]

// string list

["apple", "banana", "cherry"]

// nested list

[["list"], "of", [["lists"]]]

They both will return the list itself.

Expression:

[1, "two", [["three","four"],5],"six"]

Result:


```
[
  1,
  "two",
  [
    [
      "three",
      "four"
    ],
    5
  ],
  "six"
]
```

Expressions

In this lesson, we will explore various list expressions in FEEL. We will cover:

- Getting an element
- Filtering elements
- some and every expressions

By practicing these expressions, you will become proficient in handling list manipulations in your decision models using FEEL.

Get element

To access an element in a list, you use the index notation.

1-based indexing

The **index** of a list starts at 1. In other languages, the index starts at 0.

// Syntax

list[index]

Get the first element

["Mercury", "Venus", "Earth", "Mars"][1]

Result: "Mercury"

Get the fifth element

["Mercury", "Venus", "Earth", "Mars"][5]

Result: null

Get the element with position 0

["Mercury", "Venus", "Earth", "Mars"][0]

Result: null

Negative index

If the **index is negative**, it starts counting the elements **from the end of the list**. The last element of the list is at index -1

Get the last element

["Mercury", "Venus", "Earth", "Mars"][-1]

Result: "Mars"

Get the element with position -5

["Mercury", "Venus", "Earth", "Mars"][-5]

Result: null

Filter elements

You can filter a list based on a **condition** using a filter expression.

Info

While filtering, the current element is assigned to the variable item

// Syntax

list[condition]

Get the elements greater than 2

[1,2,3,4][item > 2]

Result: [3,4]

Get the elements greater than 10

[1,2,3,4][item > 10]

Result: []

Some

Checks if **at least one element** in the list satisfies the condition.

// Syntax

some list satisfies condition

Check if there's an element greater than 20

some x in [10,20,30] satisfies x > 20

Result: true

Check if at least one element of the first list is greater than an element of the second list

some x in [3500,3000], y in [1200,3100] satisfies x > y

Result: true

The expression returns true if any pair satisfies the condition x > y. In this case, since three pairs satisfy the condition (3500 > 1200; 3500 > 3100; 3000 > 1200), the expression evaluates to true.

Every

Checks if **all the elements** in the list satisfies the condition.

// Syntax

every list satisfies condition

Check if all the elements are greater than 5

every x in [10,20,30,40] satisfies x > 5

Result: true

Check if every elements of the first list are greater than every element of the second list

every x in [3500,3000], y in [1200,3100] satisfies x > y

Result: false

The expression returns true if all pairs satisfy the condition x > y. In this case, since three pairs satisfy the condition (3500 > 1200; 3500 > 3100; 3000 > 1200), but the last one does not satisfy the condition (3000 > 3100), the expression evaluates to false.

Number List Functions

Sum and Product

The sum() and theproduct() functions in 🔗 FEEL used in Camunda are used to calculate the **sum** and the **product** of the elements in a list, respectively.

Basic Concepts

// Syntax

sum(list)

product(list)

Calculate the sum of different numbers

sum([1, 2.5, 3.5, -3])

Result: 4

Calculate the product based on a condition

product([1, 2, 3, 4, 5][item > 2])

Result: 60

Mean, Median and Standard Deviation

The mean() , themedian(), and the stddev() functions in 🔗 FEEL used in Camunda are used to calculate the **average** (arithmetic mean), the **median** (middle value), and the **standard deviation** (the amount of variation or dispersion) of the elements in a list, respectively.

Basic Concepts

// Syntax

mean(list)

median(list)

stddev(list)

Calculate the mean of different numbers

mean([1, 2.5, 0.5, -4])

Result: 0

Calculate the median

median([1, 2, 3, 4, 5, 6])

Result: 3.5

Calculate the standard deviation

stddev([1, -2, 3, -4, 5])

Result: 3.646916505762094

Mode

The mode() function in 🔗 FEEL used in Camunda is used to determine the number(s) that **appear most frequently** in a list. If there are multiple numbers with the same highest frequency, it returns all of them. It returns a list of elements.

Basic Concepts

// Syntax

mode(list)

Find the most frequent number

mode([1, 2, 2, 3, 4])

Result: [2]

Find the mode when some numbers have the same highest frequency

mode([1, 1, -2, -2, 3, 4])

Result: [1, -2]

General List Functions

Min and Max

The min() and the max() functions in 🔗 FEEL used in Camunda are used to find the **smallest** and **largest** values in a list, respectively.

All elements in list should have the **same type** and be **comparable**.

Basic Concepts

// Syntax

min(list)

max(list)

Find the smallest number

min([0, -10, -20, 5, 15])

Result: -20

Find the latest dates

max([date("2023-01-01"), date("2022-06-15"), date("2024-03-10")])

Result: "2024-03-10"

List Contains

The list contains() function in 🔗 FEEL used in Camunda is used to check if a list contains a specific element. It returns a boolean value.

Basic Concepts

// Syntax

list contains(list, element)

Check if a number is in a list of numbers

list contains([1, 2, 3, 4, 5], 3)

Result: true

Count

The count() function in 🔗 FEEL used in Camunda is used to determine the number of elements in a list. It returns a number.

Basic Concepts

// Syntax

count(list)

Count the number of elements in a list of numbers

count([1, 2, 3, 4, 5])

Result: 5

Counting elements based on a condition

count([5, 10, 15, 20, 25][item < 20])

Result: 3

All and Any

The all() and the any() functions in 🔗 FEEL used in Camunda are **contrary to each other**:

- all(): If any boolean element in the list is false, it returns false.
  If the given list is **empty**, it returns true.
- any(): If any boolean element in the list is true, it returns true.
  If the given list is **empty**, it returns false.

Otherwise, they return null.

Basic Concepts

// Syntax

all(list)

any(list)

Check if an element is false

all([true,false,1,"Mars",null])

Result: false

Check if an element is true

any([true,false,1,"Mars",null])

Result: true

Sublist

The sublist() function in 🔗 FEEL used in Camunda is used to **extract a portion of a list** based on specified **start postion** and **lenght** indexes.

Basic Concepts

// Syntax

sublist(list, start position, length)

Returns a partial list of the given list starting at **start position**, of a given **length** (optional).

Reminder

The start position starts at the index 1. The **last position** is -1.

Get the elements from position 2

sublist(["Mars","Earth","Mercury"], 2)

Result: ["Earth","Mercury"]

Get the next 2 elements from position 1

sublist(["Mars","Earth","Mercury"], 1, 2)

Result: ["Mars","Earth"]

Append, Insert Before and Remove

In this lesson, we will cover the append(), insert before(), and the remove() functions in 🔗 FEEL used in Camunda.

- append(): Returns the given list with all items appended
  The parameter items can be a *single* element or a *sequence* of elements
- insert before(): Returns the given list with newItem inserted at position
- remove(): Returns the given list **without** the element at position

Basic Concepts

// Syntax

append(list, items)

insert before(list, position, newItem)

remove(list, position)

Append elements

append(["onions","tomatoes"],"garlic","pepper")

Result: ["onions", "tomatoes", "garlic", "pepper"]

Insert before an element

insert before(["onions","tomatoes"], 2, "garlic")

Result: ["onions", "garlic", "tomatoes"]

Remove an element

remove(["onions","tomatoes"],2)

Result: ["onions"]

Reverse and Index Of

In this lesson, we will cover the reverse() and the index of() functions in 🔗 FEEL used in Camunda.

- reverse(): Returns the given list in **reverse** order
- index of(): Returns a list with the **index(es)** of the occurrence(s) of a specified element in the list.
  If the element is not found, it returns an **empty list** []

Basic Concepts

// Syntax

reverse(list)

index of(list, element)

Reverse the elements of a list

reverse([10, 20, 30, 40, 50])

Result: [50, 40, 30, 20, 10]

Find the index of an element

index of([10, 20, 30, 40, 50], 30)

Result: [3]

Union, Distinct Values, Duplicate Values

In this lesson, we will cover the union(), distinct values(), and the duplicate values() functions in 🔗 FEEL used in Camunda.

- union(): Returns a list containing all the elements from the input lists, **without any duplicates**
- distinct values(): Returns a list containing only the **distinct (unique) values** from the input list
- duplicate values(): Returns a list containing only the **duplicate values** from the input list.

Basic Concepts

// Syntax

union(lists)

distinct values(list)

duplicate values(list)

Union of two lists of numbers

union([1, 2, 3, 4],[3, 4, 5])

Result: [1, 2, 3, 4, 5]

Distinct values in a list of numbers

distinct values([1, 2, 2, 3, 4, 4, 5])

Result: [1, 2, 3, 4, 5]

Duplicate values in a list of numbers

duplicate values([1, 2, 2, 3, 4, 4, 5])

Result: [2,4]

Sort

The sort() function in 🔗 FEEL used in Camunda defines the sorting order using a **custom precedes function**. This advanced feature allows you to sort lists based on custom criteria.

Basic Concepts

// Syntax

sort(list, precedes: function<(Any, Any) -> boolean>)

- **list**: The list to be sorted.
- **precedes**: A custom function that defines the sorting order. It takes two arguments and returns a boolean indicating whether the first argument should precede the second.

Sorting numbers in descending order

sort([3, 1, 4, 5, 2], function(x,y) x > y)

Result: [5, 4, 3, 2, 1]

Flatten, Concatenate and String Join

In this lesson, we will cover the flatten(), concatenate(), and the string join() functions in 🔗 FEEL used in Camunda. These functions help in manipulating lists and strings for various purposes in decision models.

- flatten(): takes a nested list and **merges the elements into a single list**
- concatenate(): Concatenates **multiple lists** into a single list
- string join(): Joins the strings of a list into a **single string**, with an optional delimiter between elements and/or an optional prefix or sufix

Basic Concepts

// Syntax

flatten(nested list)

concatenate(lists)

string join(list, delimiter, prefix, suffix)

Flatten a nested list

flatten([[1, 2, 3], [4, 5], [6, 7, 8]])

Result: [1, 2, 3, 4, 5, 6, 7, 8]

Concatenate multiple lists

concatenate([1, 2], [3, 4], [5, 6])

Result: [1, 2, 3, 4, 5, 6]

Join a list of strings with a delimiter

string join(["Camunda", "FEEL", "is", "great"], " ")

Result: "Camunda FEEL is great"

Review

Through this course, you have learned how to evaluate **FEEL List** data types, expressions and functions used in Camunda.

What Did I Learn?

You should now be able to:

- Understand the basic list data type
- Operate with various list expressions:
    - **Get**
    - **Filter**
    - **Some**
    - **Every**
- Use various number list functions:
    - min() and max()
    - sum() and product()
    - mean(), median(), and stddev()
    - mode()
- Use various general list functions:
    - list contains()
    - count()
    - all() and any()
    - sublist()
    - append(), insert before(), and remove()
    - reverse() and index of()
    - union(), distinct values(), and duplicate values()
    - sort()
    - flatten(), concatenate(), and string join()