FEEL - Control Flow and Advanced Expressions

What Will I Learn?

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

- Construct and understand complex conditional logic in FEEL and use logical operators effectively
- Use loops and iteration for complex data manipulation in FEEL and apply these skills in practical scenarios
- Create custom functions to encapsulate complex logic and apply these functions in various business scenarios

Control Flow

Advanced Conditionals

By mastering advanced conditionals, you will be better equipped to handle complex decision logic in your business processes using FEEL.

Recap of Basic Conditionals

Before diving into advanced conditionals, let's briefly recap the basics.

A simple conditional in FEEL evaluates a condition and returns a result based on whether the condition is true or false.

Syntax:

```
if condition then result1 else result2
```

Example:

```
if order.totalAmount > 50 then "Free Shipping" else "Standard Shipping"
```

Nested Conditionals

Nested conditionals are conditionals within conditionals. They allow for more complex decision logic by evaluating multiple conditions in a hierarchical manner.

Syntax

```
if condition1 then
  if condition2 then result1 else result2
else
  if condition3 then result3 else result4
```

Example

Determine the discount based on customer type and order amount:

```
if customer.isPremiumMember then
  if order.totalAmount > 100 then 20 else 10
```

```
else
  if order.totalAmount > 100 then 5 else 0
```

Using Logical Operators

Logical operators (and and or) allow for combining multiple conditions into a single conditional statement.

AND Operator

Evaluates to true if all conditions are true.

```
if condition1 and condition2 then result1 else result2
```

Example

```
if order.totalAmount > 50 and customer.isPremiumMember then "Free Shipping" else
"Standard Shipping"
```

OR Operator

Evaluates to true if at least one condition is true.

```
if condition1 or condition2 then result1 else result2
```

Example

```
if order.totalAmount > 50 or customer.isPremiumMember then "Free Shipping" else
"Standard Shipping"
```

not()

Boolean functions like not() returns the logical negation of the given value.

NOT Operator

Reverses the boolean value of a condition.

```
if not(condition) then result1 else result2
```

Example

```
if not(customer.isPremiumMember) then "Standard Shipping" else "Free Shipping"
```

Loops and Iterations

In FEEL, the primary looping construct is the for loop, which can be used to iterate over a collection of items, performing operations on each item. This is particularly useful for processing lists.

Using for Loops

Syntax:

```
for element in list return expression
```

Example: Suppose you have a list of prices and you want to apply a 10% discount to each price:

```
for price in [100, 200, 300] return price * 0.9
```

Output:

```
[90, 180, 270]
```

You can also use multiple variables in a for loop:

```
for x in [1, 2, 3], y in [4, 5, 6] return x + y
```

Output:

```
[5, 6, 7, 6, 7, 8, 7, 8, 9]
```

Note

FEEL does not have a built-in filter function, but you can achieve a similar effect using conditional expressions within a loop.

Custom Functions

Custom functions in FEEL allow you to encapsulate complex logic into reusable blocks. This helps in simplifying expressions, improving readability, and promoting reuse across different decision tables and business rules.

Defining Custom Functions

To define a custom function in FEEL, you specify the function name, parameters, and the expression that the function evaluates.

Syntax:

```
function(parameter1, parameter2, ...) expression
```

Example: Define a function to calculate the square of a number:

```
{
  square: function(x) x * x
}
```

Using Custom Functions

Once defined, you can use custom functions just like any other built-in function.

Example: Define a custom function to calculate the area of a circle given its radius:

```
{
  pi: 3.14159,
  areaOfCircle : function(radius) (pi * radius * radius),
  result: areaOfCircle(3)
}.result
```

Output:

```
28.27431
```

You can also define functions that take multiple parameters. For example, a function to calculate the sum of two numbers:

```
{
  sumTwoNumbers: function(a, b) a + b
}
```

Example: Use the sumTwoNumbers function to add 3 and 5:

```
{
  sumTwoNumbers: function(a, b) a + b,
  result: sumTwoNumbers(3,5)
}.result
```

Output:

```
8
```

Challenge

Instructions

Objective

In this challenge, you will use your knowledge of FEEL expressions to analyze and manipulate a context containing data about a process instance.

Here is the context you will be working with:

```
{
  "processInstance": {
      "id": "P123456789",
      "startDate": "2023-01-01T09:00:00Z",
      "tasks": [
        {"id": "T2", "name": "Gather Requirements", "duration": 120, "completed": true},
        {"id": "T6", "name": "Deployment", "duration": 90, "completed": false},
        {"id": "T4", "name": "Implement Solution", "duration": 240, "completed": false},
        {"id": "T3", "name": "Design Solution", "duration": 180, "completed": false},
        {"id": "T5", "name": "Testing & QA", "duration": 180, "completed": false},
        {"id": "T1", "name": "Initial Assessment", "duration": 60, "completed": true}
      ],
      "variables": {
      "budget": 10000,
      "expenses": 4500,
      "stakeholders": [
        {"name": "Alice", "role": "Sponsor"},
        {"name": "Bob", "role": "Product Owner"},
        {"name": "Charlie", "role": "Lead Developer"}
      ]
    }
  }
}
```

Tasks

Calculate Total Duration of Completed Tasks

Write a FEEL expression to calculate the total duration of all completed tasks. Update the processInstance object by adding a new element totalCompletedDuration with the calculated value.

Hints:

- Start by filtering the tasks that are completed
- Get the duration of the completed tasks
- Finally sum all completed tasks duration and update the processInstance object with a new element with the calculated value

Check Budget Status

Write a FEEL expression that compares the current expenses against the budget and determines the budget status as "Under Budget", "On Budget", or "Over Budget". Update the processInstance variables with a new key-value pair budgetStatus.

Hints:

- Start by creating the condition for the expenses to be less, equals, and greater than the budget and return the text value according to the status
- Finally update the processInstance object with a new element with the calculated value

Identify and Sort the Next Task to Complete

Write a FEEL expression to find the first uncompleted task in the process sorted by id in alphabetical order. Update the processInstance object by adding a new element nextTask which should be the name of the next task to complete.

Hints:

- Start by filtering the tasks that are not completed
- Sort the filtered list by the task id in alphabetical order
- Get the name of the first incompleted task
- Finally update the processInstance object with a new element with the calculated value

Custom Function for Stakeholder Roles

Create a custom FEEL function named stakeholderRoles that takes the stakeholders list and returns a list of roles. Use this function to update the processInstance object by adding a new property rolesList that contains the list of roles from the stakeholders.

Hints:

- Start by creating a custom function to loop the stakeholders and return the list of roles
- Call the function with the argument's list of stakeholders to obtain the listOfRoles
- Finally update the processInstance object with a new element with the calculated value

Solution

Calculate Total Duration of Completed Tasks

Expression:

```
context put(processInstance, “totalCompletedDuration”,
sum(processInstance.tasks[completed = true].duration))
```

Result:

```
{
"variables": {
  "stakeholders": [
    {"name": "Alice", "role": "Sponsor"},
    {"name": "Bob", "role": "Product Owner"},
    {"name": "Charlie", "role": "Lead Developer"}
  ],
  "expenses": 4500,
  "budget": 10000
},
"id": "P123456789",
"tasks": [
  {"id": "T2", "name": "Gather Requirements", "duration": 120, "completed": true},
  {"id": "T6", "name": "Deployment", "duration": 90, "completed": false},
  {"id": "T4", "name": "Implement Solution", "duration": 240, "completed": false},
  {"id": "T3", "name": "Design Solution", "duration": 180, "completed": false},
  {"id": "T5", "name": "Testing & QA", "duration": 180, "completed": false},
  {"id": "T1", "name": "Initial Assessment", "duration": 60, "completed": true}
],
"totalCompletedDuration": 180,
"startDate": "2023-01-01T09:00:00Z"
}
```

Check Budget Status

Expression:

```
context put(processInstance.variables,”budgetStatus”,if processInstance.variables.expenses
< processInstance.variables.budget then “Under Budget” else if
processInstance.variables.expenses = processInstance.variables.budget then “On Budget”
else “Over Budget”)
```

Result:

```
{
  "stakeholders": [
    {
    "role": "Sponsor",
    "name": "Alice"
    },
    {
    "role": "Product Owner",
    "name": "Bob"
    },
    {
    "role": "Lead Developer",
    "name": "Charlie"
    }
  ],
  "expenses": 4500,
  "budget": 10000,
  "budgetStatus": "Under Budget"
}
```

Identify and Sort the Next Task to Complete

Expression:

```
context put(processInstance, "nextTask", sort(processInstance.tasks[not(completed)],
function(x,y) x.id< y.id)[1].name)
```

Result:

```
{
  "variables": {
    "stakeholders": [
      {"name": "Alice", "role": "Sponsor"},
      {"name": "Bob", "role": "Product Owner"},
      {"name": "Charlie", "role": "Lead Developer"}
    ],
    "expenses": 4500,
    "budget": 10000
  },
  "id": "P123456789",
  "tasks": [
    {"id": "T2", "name": "Gather Requirements", "duration": 120, "completed": true},
    {"id": "T6", "name": "Deployment", "duration": 90, "completed": false},
    {"id": "T4", "name": "Implement Solution", "duration": 240, "completed": false},
    {"id": "T3", "name": "Design Solution", "duration": 180, "completed": false},
    {"id": "T5", "name": "Testing & QA", "duration": 180, "completed": false},
    {"id": "T1", "name": "Initial Assessment", "duration": 60, "completed": true}
  ],
  "nextTask": "Design Solution",
  "startDate": "2023-01-01T09:00:00Z"
}
```

Custom Function for Stakeholder Roles

Expression:

```
{
  "getList": function(stakeholders) for s in stakeholders return s.role,
  "listOfRoles": getList(processInstance.variables.stakeholders),
  "processInstanceUpdated": context put(processInstance, "rolesList", listOfRoles)
}.processInstanceUpdated
```

Result:

```
{
  "rolesList": [
    "Sponsor",
    "Product Owner",
    "Lead Developer"
  ],
  "variables": {
    "stakeholders": [
      {"name": "Alice", "role": "Sponsor"},
      {"name": "Bob", "role": "Product Owner"},
      {"name": "Charlie", "role": "Lead Developer"}
    ],
    "expenses": 4500,
    "budget": 10000
  },
  "id": "P123456789",
  "tasks": [
    {"id": "T2", "name": "Gather Requirements", "duration": 120, "completed": true},
    {"id": "T6", "name": "Deployment", "duration": 90, "completed": false},
    {"id": "T4", "name": "Implement Solution", "duration": 240, "completed": false},
    {"id": "T3", "name": "Design Solution", "duration": 180, "completed": false},
    {"id": "T5", "name": "Testing & QA", "duration": 180, "completed": false},
    {"id": "T1", "name": "Initial Assessment", "duration": 60, "completed": true}
  ],
  "startDate": "2023-01-01T09:00:00Z"
}
```

Review

Through this course, you have learned how to evaluate FEEL Control Flow and Advanced Expressions in Camunda.

What Did I Learn?

You should now be able to:

- Construct and understand complex conditional logic in FEEL and use logical operators effectively
- Use loops and iteration for complex data manipulation in FEEL and apply these skills in practical scenarios
- Create custom functions to encapsulate complex logic and apply these functions in various business scenarios