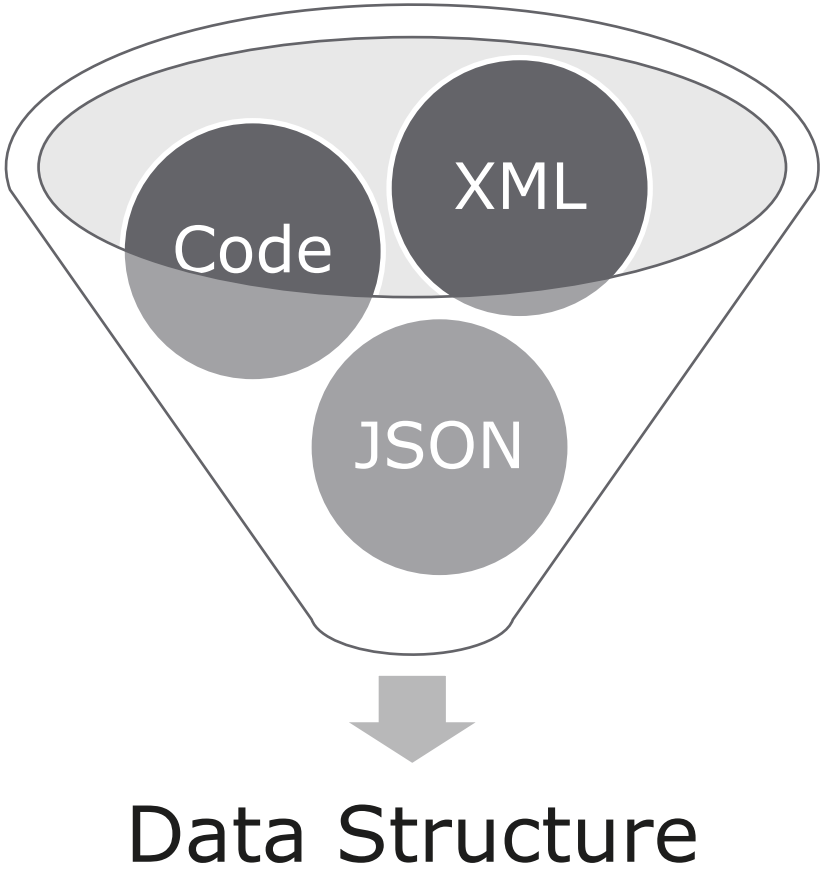
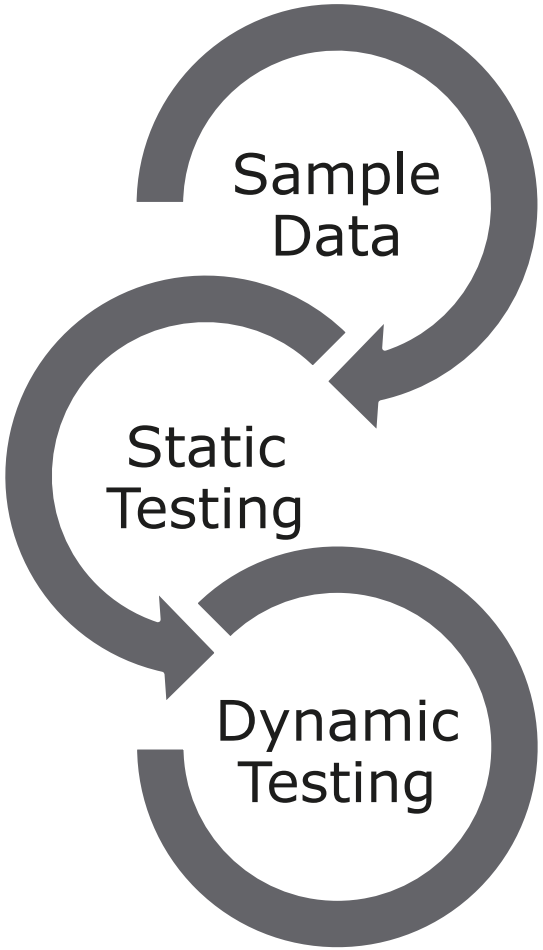
Module 1: DataWeave Fundamentals—Review++

Goal

Inputs and outputs of DW



Development process



At the end of this module, you should be able to

- Use DataWeave as learned in *Development Fundamentals* course
- Configure metadata for DataWeave transformation input and output
- Set example input for DataWeave transformations

Reviewing DataWeave fundamentals

Why DataWeave?

- DataWeave is a simple JSON-like functional programming language to transform and query data
- Easy to write, easy to maintain, and capable of supporting simple to complex mappings between any data types
  - Out of the box support for many popular file and data types
    - XML, JSON, Java, CSV, EDI, Excel, fixed-width files, flat files, and COBOL copybook

Example DataWeave transformation expression

The **header** contains directives
- High level info about the transformation
- Variables, and functions
- Divided by the body separator ---

Input


```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:listAllFlightsResponse xmlns:ns2="
    <return>
        <airlineName>Delta</airlineName
        <code>A1B2C3</code>
        <departureDate>2015/03/20</depa
        <destination>SFO</destination>
        <emptySeats>40</emptySeats>
        <origin>MUA</origin>
        <planeType>Boing 737</planeType
        <price>400.0</price>
```

Transform

```
1  %dw 2.0
2  output application/java
3  ns ns0 http://soap.training.mulesoft.com/
4  type Currency = String {format: "###.00"}
5  type Flight = Object {class: "com.mulesoft.training.Flight"}
6  ---
7  payload.ns0#findFlightResponse.*return
8
9  map ((return , indexOfReturn) -> {
10     airlineName: return.airlineName,
11     availableSeats: return.emptySeats as Number,
12     departureDate: return.departureDate as Date {format: "yyyy/MM/dd"}
13     as String {format: "MMM dd, yyyy"},
14     destination: return.destination,
15     flightCode: return.code,
16     origination: return.origin,
17     planeType: upper (return.planeType replace /(Boing)/ with "Boeing"),
18     price: return.price  as Number as Currency
19 } as Flight
20 )
```

Output

Preview

Name	Value
root : *LinkedHashMap*	
flights : *ArrayList*	
[0] : *Flight*	
airlineName : *String*	
availableSeats : *Integer*	40
departureDate : *String*	Oct 20, 2015
destination : *String*	SFO
flightCode : *String*	
origination : *String*	
planeType : *String*	BOEING 737
price : *Double*	400.0
[1] : *Flight*	

The **body** contains a DataWeave expression that generates the output data

What characterizes DataWeave as a **functional programming language**?

- DataWeave is a functional programming language
- Functional programming languages only have expressions
  - There are NO statements in DataWeave!
  - Expression have no side-effects!
- Functions are values
  - Anywhere you use an integer you can use a function
- Functions with exactly two arguments
  - Are applied/invoked using an infix syntax
- Infix syntax promotes functional program composition
  - through expression chaining

```
payload orderBy $.price distinctBy $
```

DataWeave documentation

- https://docs.mulesoft.com/mule-user-guide/v/4.1/dataweave

https://docs.mulesoft.com/mule-runtime/4.1/dataweave

MuleSoft Product Solutions Services Resources Company Developers Partners

Mule Runtime 4.1

- Mule Runtime Overview
- Mule Installation
- Mule 4 for Mule 3 Users
- Mule Apps
- Mule Events
- Core Components
- Strategies and Patterns for Mule Apps
- Mule Maven Plugin
- DataWeave Language
  - DataWeave Scripts
  - DataWeave Selectors
  - Data Formats Supported by DataWeave

DataWeave Language

DataWeave is the MuleSoft expression language for accessing and transforming data received through a Mule app. DataWeave is tightly integrated with Mule runtime, which runs the scripts and expressions in your Mule app.

DataWeave acts on data in the Mule event. Most commonly, you use it to access and transform data in the message payload. For example, after a component in your app retrieves data from one system, you can use DataWeave to modify and output selected fields in that data to a new data format, then use another component in your app to pass on that data to another system.

The simple example below demonstrates a few important features of the DataWeave language:

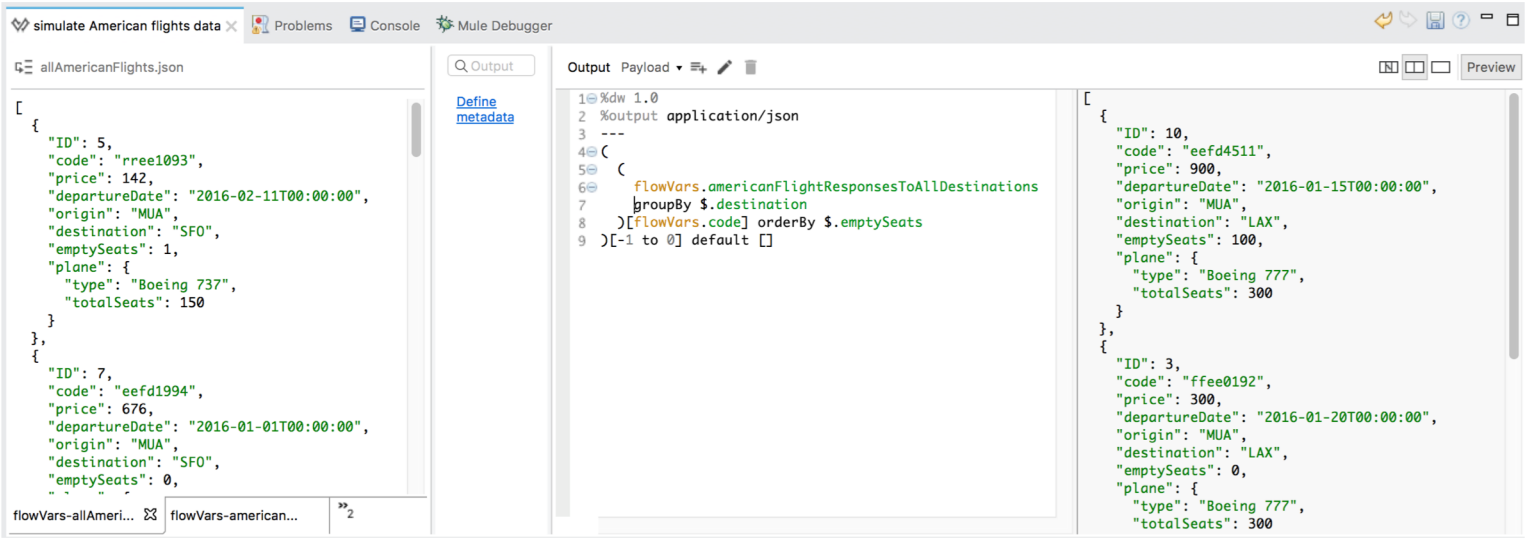
- Data transformation from one format to another. The example transforms `application/json` input to `application/xml` output.
- Data selectors that access fields within an input data structure. The script selects keys in a JSON object from the message payload input (for example, with `payload.title`) and then outputs their values.
- Use of a core DataWeave function on the value of an input field `upper(payload.author)`.

Walkthrough 1-1: New project with sample data

- Create a new project
- Copy sample data in the project structure
  - A CSV file containing records describing airports
  - A JSON file containing a single object describing a flight
  - An XML file containing multiple records describing flights

Walkthrough 1-2: Fundamentals Review++

- Review object/array creation
- Field accessors
- Array accessors
- String concatenation
- Expression chaining
- Conditional expressions
- Ranges
- `typeOf`, `sizeOf`, `is`, `contains`
- Review XML to JSON transformations
- Review JSON to XML transformations



Summary

Summary

- DataWeave is a functional programming language with JSON-like syntax
- DataWeave supports XML, JSON, Java, CSV, EDI, fixed width, flat file, and COBOL copybook out-of-the-box
- Inputs are combined with the DW expression to build a data structure
- The data structure must be agreeable with the requested output