# Contents

# Module 1

# Fundamentals – Review++

## WT 1-1 Import a basic Mule project into Anypoint Studio

### Import the starter project

1. Start Anypoint Studio
2. Create a new workspace
3. Import the `apdw2-flights-starter.jar` project under the `studentFiles/mod01`

### Create new project

4. Create a new project
   *Creating a new project and copying only the files you minimally need for the class helps in containing the "noise" that is introduced with starter project. Additionally, there is the extra benefit of not having to deal with students who are having compilation issues with the starter project.*
5. Create a new project and call it dataweave
6. From the apdw2-flights-starter copy the following files over to the new project:
   (a) `src/main/resources/airportInfoTiny.csv` to `src/main/resources`
   (b) `src/main/resources/examples/mockdata/deltaSoapResponsesToAllDestinations.xml` to `src/test/resources`
   (c) `src/test/resources/flight-example.json` to `src/test/resources`

## WT 1-2 Fundamentals review++

*In this WT the goal is to attempt (I am saying attempt because often enough we have participants who don't meet the prerequisites) to bring everyone at the same level by (1) reviewing fundamentals and (2) illustrating features of DW that we will be using throughout the class*

### Create the flow, set the metadata

1. Rename the `dataweave.xml` to `mod1.xml`
2. Create a new flow named `mod1-review++`
   *The reason for prefixing the flow name with the name of the flow is a best-practice one. Such a convention will improve the readability of your flows by identifying the Mule Configuration file a flow is defined under by just looking at a Flow Reference's display name.*
3. Drop a DW (aka `Transform Message`) to the process area of the flow
4. Define the payload input metadata to the `src/test/resources/flight-example.json`, set the name of the type to `flight_json`
5. Edit the sample data
6. Turn on the preview
7. Change the output to JSON

## Construction

8. What are the semantics of {} in DW?
   (a) Object creation
9. What are the semantics of `[]` in DW?
   (a) Array creation

## Fields

10. Three different ways of accessing the field `airline` out of the `payload`. What are they?
    (a) `payload.airline`
    (b) `payload["airline"]`
    (c) `payload[0]`

    *Let me let you in a secret: Objects internally are represented as arrays—field access is a façade*

11. Why DW stores objects as arrays?
    (a) Because DW is the only language I know of that allows the creation of objects with duplicate field names...

```
{
  a: 1,
  a: 2,
  a: 3
}
```

    ... and the only way I can access the second and third field is through an index access. But now we have more questions that need to be answered.
    (b) Why would a language allow for such a feature? That is duplicate fields within an object.
        i. Because of XML, how else you expect to be able to generate XML with tags that repeat:

```
%dw 2.0
output application/xml
---
"as": {
  a: 1,
  a: 2,
  a: 3
}
```

## String concatenation

12. Two ways to concatenate strings
    (a) `"The flight is operated by " ++ payload.airline`
    (b) `"The flight is operated by ${payload.airline}"`
13. You have to be careful that the expression inside the `${}` returns a string, otherwise you will be getting type missmatch errors.

## Conditional expressions

14. `if then else` conditional
    (a) `if (true) 1 else 0`
    (b) `if (false) 1 else 0`
15. Nullity conditional
    (a) lstinlinenull default "Other value"
    (b) lstinline"The value" default "Other value"

16. Conditional elements

    (a) Objects

```
{
  a: 1,
  (b: 2) if (true),
  (c: 3) if (false)
}
```

    (b) Arrays

```
[
  1,
  (2) if (true),
  (3) if (false)
]
```

## Array access and Ranges

17. Array access

    (a) `[2,6,4,1,7][0]` evaluates to `2`

    (b) `[2,6,4,1,7][-1]` evaluates to `7`

18. Ranges

    (a) `0 to 5` evaluates to the `[0,1,2,3,4,5]` array

    (b) `5 to 0` evaluates to the `[5,4,3,2,1,0]` array

19. Ranges, Arrays, and Strings

    (a) `[2,6,4,1,7][1 to -2]` evaluates to the `[6,4,1,7]` sub-array

    (b) `[2,6,4,1,7][-1 to 0` reverses the array

    (c) `payload.airline[-3 to -1]` evaluates to the last characters in the string

    (d) `payload.airline[-1 to 0]` reverses the string

## Common functions

20.

## Expression chaining

## Transform XML to JSON

## Transform JSON to XML

## Module 2

# Variables, Functions, Modules

**WT 2-1 Organize DataWeave code with variables and functions**

**WT 2-2 Reuse DataWeave transformations**

**WT 2-3 Create and use DataWeave modules**

## Module 3

# Defensive programming

## Module 4

# Operating on Arrays and Objects

## Module 5

# The Arrays and Objects Modules

## Module 6

# Flights and Airports

# Module 7

# Recursion