LAB- Dw modules, orderBy,Filter and function chaining

STEP 1

You will be working on the previous project only.

We want to write reusable functions and variables in custom dwl modules

Create a folder with name mymodules under src/main/resources

Inside modules folder, create a File with name CustomFunctions.dwl

Cut the function **getOfferPrice** and variable **vGetOfferPrice** from the header part of the earlier dwl and paste it in CustomFunctions.dwl as shown below:

transformerdemo | *CustomFunctions.dwl

```
fun getOfferPrice(price,discountPercentage)
            = price*(100-discountPercentage)/100


var vGetOfferPrice = (price,discountPercentage) ->
                          price*(100-discountPercentage)/100
```

Now import and call the function as shown below:

```
%dw 2.0
output application/xml
import  mymodules::CustomFunctions

var conversionRate=74


---
product @(pid:payload.productId):{

    productName:payload.name,
    originalPrice: payload.originalPrice,
    offer:{
        discountPercentage: payload.offer.discountPercentage default 0,
        offerPrice: CustomFunctions::getOfferPrice(payload.originalPrice,
                    payload.offer.discountPercentage),

        offerValidUntil:payload.offer.offerValidUntil as Date
                                    },
    image1 :payload.images[0]
}
```

Change import as shown below to assign an alias to your function:

```
import getOfferPrice as gop,vGetOfferPrice as vgop from mymodules::CustomFunctions
```

Now change the dwl to call the function using the alias as shown below:

```
offerPrice: gop(payload.originalPrice,payload.offer.discountPercentage),
```

As dataweave is a functional language, we can pass functions as arguments to another function

Define a function as shown below:

```
fun formatData(myinput , formatter) =formatter(myinput)
```

Now change the body expression to

```
formatData("SivaPrasad",lower)
```

Observe that output is is in lower case.

Now change body expression to `formatData("SivaPrasad",upper)` and observe the result in upper case

We want to try various String functions present in dw::core::Strings module.

So, import dw::core::Strings by writing `import dw::core::Strings`

Now change the body expression to `formatData("SivaPrasad",Strings::camelize)` and observe the result

Now change the body expression to `formatData("SivaPrasad",Strings:: capitalize)` and observe the result

Now change the body expression to `formatData("product",Strings::pluralize)` and observe the result

Now change the body expression to `formatData("product",Strings::singularize)` and observe the result

Now change the body expression to `formatData("Siva Prasad",Strings::underscore)` and observe the result

Now change the body expression to `formatData(1, Strings::ordinalize)` and observe the result

Now change the body expression to `formatData("Siva Prasad",Strings::dasherize)` and observe the result

We can also call our function as shown below:

```
"Siva Prasad" formatData Strings::dasherize
```

We can strongly type function arguments and functions like below:

```
fun formatData(myinput:String , formatter: (Number)->String) =formatter(myinput)
```

in the body expression vall formatData using

`formatData("Siva Prasad",Strings::dasherize)`

You should observe an error because `Strings::dasherize` function takes String as argument

Now modify the function as shown below so that errors will disappear

```
fun formatData(myinput:String , formatter: (String)->String) =formatter(myinput)
```

Try to use Strings::wrapWith, Strings::leftPad

Chaining Functions

There are some functions which take only one argument and they cannot be chained.

What do I mean by function chaining?

See this below example:

```
[1,5,3,2]
filter ($ mod 2) == 1
orderBy $
```

We are able to chain filter and orderBy functions because they take 2 arguments

Let us try to chain lower.

Change the body expression to `"Siva Prasad"_lower`

You should observe an error because lower function takes only one argument

Now let us create a function which allows us to use lower function in chained way.

```
fun chain(myinput, myfunction) =
                        myfunction(myinput)
```

Now change the body expression as below.

```
"Siva Prasad" chain lower
```

You should see the result with out error.

Now change body expression to

```
{one:"One", two:"Two"} chain (x) -> (x.one ++ ' ' ++ x.two)
```

You should observe the below result: `"One Two"`

Change the body expression to

```
upper( {one:"One", two:"Two"} chain (x) -> (x.one ++ ' ' ++ x.two) )
```

We can achieve the same result using below expression:

```
{one:"One", two:"Two"} chain (x) -> (x.one ++ ' ' ++ x.two)  chain upper
```

Above expression is same as

```
chain(
chain( {one:"One", two:"Two"}, (x) -> (x.one ++ ' ' ++ x.two) ),
upper
)
```

This is the end of the Exercise