

Shangqing Hu – SEC01 (NUID 001374342)

Big Data System Engineering with Scala  
Spring 2023  
Assignment No.5(Functional-composition)



## -List of Tasks Implemented

his assignment builds on what we already did for the Movie database in order to test what you learned recently about functional composition, *Option*, *Try*, for comprehensions, etc.

I had a lot of fun with this assignment, and a little frustration at times. But that's to be expected any time you are doing something challenging. If you follow the mantra Simple, Obvious, Elegant, you won't have too much trouble. As noted above, I've actually done the really hard stuff for you (that's all in the *Movie.scala* file).

OK, to business...

All the work you are going to do is in the *asstfc* package. *Ingest* hasn't changed, but *Movie* (and *MovieSpec* - also *inasstfc*) have.

There are 13 TODOs in *Function.scala* and 2 TODOs in *Movie.scala*. You can get these from the class repo (see *Course Material/Resources/Class Repository*), the module name for this assignment is *assignment-functional-composition*.

Did you find it at all bothersome (in Assignments 1 and 2) that we didn't use *Try* (or *Option*) to deal with some of the *String => Int* conversions? Some of those invocations of *toInt* might easily have thrown exceptions. Part of the problem was that it was too difficult to do, given your knowledge of Scala at the time.

But now, you know all about functional composition, for comprehensions and monads.

The basic problem that I had to solve is that some of the *apply* functions (for example in *Format*) have a mixture of strings, as well as *Int* and *Double* which require conversion from *String*. One of the elements of a *Movie* (viz. *Reviews*) was based on seven *Int* conversions from seven strings. That's easy: just use *map7* (!). Of course we didn't actually implement *map7* in class, but you're going to implement it in this assignment. But what about mixtures like in *Format*? That required a rather creative solution (if I may say so). Take a look at the code in the object *Format* and also in the object *Production*. First, we need to curry the *apply* method. Functions in curried form are much more tractable than functions in tupled (normal) form. Now, the rest of the code transforms that curried function into a function that can be used in a *for* comprehension. Don't worry too much if you don't understand what's going on here. But, I believe that if you take the time to figure it out, you can see what's happening.

In any case, this code uses some utilities that I placed in the *Function* object.

Now, here's an important hint for the one *TODO* that is in the *Movie.scala* file: Back in assignment 2, we had a *for comprehension* which utilized a filter (you can see that by taking a look at *Movie.scala* in *assignment-movie-database* which is unchanged). At the time, we hadn't talked about *for* comprehensions but hopefully you figured it out (it is kind of obvious).

But do you remember in a recent lecture that I said that the left hand side of a generator (in a *for comprehension*) is actually a pattern? Yes, it's a pattern just like in the case statement of a match expression. Neat, huh? So, you are going to implement the filter on the country value by using a pattern in the *for comprehension* instead of a filter (thus you will not be using the method *isKiwi* or anything like it). Do take a look at how it was implemented (using a filter) in our previous version.

But there's one tricky bit of syntax which we haven't covered yet. In Scala there can arise a certain ambiguity when you have identifiers that are the same as keywords (not allowed in Java) or, as here, you have a variable mentioned in a pattern matching context where you don't want just to match on anything (you would be "shadowing" the variable in such a case). You actually want to match on the value of the variable. For this (and the similar case where you want to use a keyword as a variable), just enclose the variable name inside back-ticks the ``` character.

## -Code

### Function.scala

```
new *
def map2[T1, T2, R](t1y: Try[T1], t2y: Try[T2])(f: (T1, T2) => R): Try[R] = for(t1 <- t1y; t2 <- t2y) yield f(t1, t2) // TO BE IMPLEMENTED

def map3[T1, T2, T3, R](t1y: Try[T1], t2y: Try[T2], t3y: Try[T3])(f: (T1, T2, T3) => R): Try[R] = for(t1 <- t1y; t2 <- t2y; t3 <- t3y) yield f(t1, t2, t3)

def map7[T1, T2, T3, T4, T5, T6, T7, R](t1y: Try[T1], t2y: Try[T2], t3y: Try[T3], t4y: Try[T4], t5y: Try[T5], t6y: Try[T6], t7y: Try[T7])
  (f: (T1, T2, T3, T4, T5, T6, T7) => R): Try[R] = for (t1 <- t1y; t2 <- t2y; t3 <- t3y; t4 <- t4y; t5 <- t5y; t6 <- t6y; t7 <- t7y) yield f(t1, t2, t3, t4, t5, t6, t7)

def lift[T, R](f: T => R): Try[T] => Try[R] = _ map f // TO BE IMPLEMENTED

def lift2[T1, T2, R](f: (T1, T2) => R): (Try[T1], Try[T2]) => Try[R] = map2(_, _)(f) // TO BE IMPLEMENTED

def lift3[T1, T2, T3, R](f: (T1, T2, T3) => R): (Try[T1], Try[T2], Try[T3]) => Try[R] = map3(_, _, _)(f) // TO BE IMPLEMENTED

def lift7[T1, T2, T3, T4, T5, T6, T7, R](f: (T1, T2, T3, T4, T5, T6, T7) => R):
  (Try[T1], Try[T2], Try[T3], Try[T4], Try[T5], Try[T6], Try[T7]) => Try[R] = map7(_, _, _, _, _, _, _)(f) // TO BE IMPLEMENTED

def invert2[T1, T2, R](f: T1 => T2 => R): T2 => T1 => R = t2 => t1 => f(t1)(t2) // TO BE IMPLEMENTED

def invert3[T1, T2, T3, R](f: T1 => T2 => T3 => R): T3 => T2 => T1 => R = t3 => t2 => t1 => f(t1)(t2)(t3) // TO BE IMPLEMENTED

def invert4[T1, T2, T3, T4, R](f: T1 => T2 => T3 => T4 => R): T4 => T3 => T2 => T1 => R = t4 => t3 => t2 => t1 => f(t1)(t2)(t3)(t4) // TO BE IMPLEMENTED

def uncurried2[T1, T2, T3, R](f: T1 => T2 => T3 => R): (T1, T2) => T3 => R = (t1, t2) => t3 => f(t1)(t2)(t3) // TO BE IMPLEMENTED

def uncurried3[T1, T2, T3, T4, R](f: T1 => T2 => T3 => T4 => R): (T1, T2, T3) => T4 => R = (t1, t2, t3) => t4 => f(t1)(t2)(t3)(t4) // TO BE IMPLEMENTED

def uncurried7[T1, T2, T3, T4, T5, T6, T7, T8, R](f: T1 => T2 => T3 => T4 => T5 => T6 => T7 => T8 => R): (T1, T2, T3, T4, T5, T6, T7) => T8 => R =
  (t1, t2, t3, t4, t5, t6, t7) => t8 => f(t1)(t2)(t3)(t4)(t5)(t6)(t7)(t8) // TO BE IMPLEMENTED
```

### Movie.scala

```
object MoviesProtocol extends DefaultJsonProtocol {
  // 20 points
  // TO BE IMPLEMENTED
  implicit val formatFormat = jsonFormat4(Format.apply)
  implicit val productionFormat = jsonFormat4(Production.apply)
  implicit val ratingFormat = jsonFormat2(Rating.apply)
  implicit val reviewsFormat = jsonFormat7(Reviews.apply)
  implicit val nameFormat = jsonFormat4(Name.apply)
  implicit val principalFormat = jsonFormat2(Principal.apply)
  implicit val movieFormat = jsonFormat11(Movie.apply)
```

```
import MoviesProtocol._
val SerializeAndDeserialize = ms.map(_._toJson.convertTo[Movie])
ms == SerializeAndDeserialize
```

## -Unit tests

Run: FunctionSpec

✓ Tests passed: 8 of 8 tests – 51ms

/Users/chubby/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java ...

Testing started at 4:26 PM ...

Test Results	Time
FunctionSpec	51ms
map2	25ms
map7	10ms
invert2	5ms
invert3	5ms
invert4	3ms
uncurried2	3ms
should work	3ms

Run: MovieSpec

✓ Tests passed: 16 of 16 tests – 402ms

/Users/chubby/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java ...

Testing started at 4:26 PM ...

Test Results	Time
MovieSpec	402ms
Name	27ms
Principal	6ms
Rating	7ms
Format	8ms
Production	7ms
Reviews	6ms
Movie.getMoviesFromCountry	201ms
should work for the sample file	201ms
Movie.testSerializationAndDeserialization	140ms
should work for the sample file	140ms