

Olasunkanmi Olayinka – SEC01 (NUID 001512266)

# Big Data System Engineering with Scala

## Fall 2022

### Assignment No. 05



## -List of Tasks Implemented

- Implement Function.scala:
  1. map2
  2. map3
  3. map7
  4. lift
  5. lift2
  6. lift3
  7. lift7
  8. invert2
  9. invert3
  10. invert4
  11. uncurried2
  12. uncurried3
  13. uncurried7
- Implement Movie.scala:
  1. MoviesProtocol
  2. testSerializationAndDeserialization

## -Code

map2:

```
12 /**  
13  * The map2 function. You already know this one!  
14  *  
15  * @param t1y parameter 1 wrapped in Try  
16  * @param t2y parameter 2 wrapped in Try  
17  * @param f function that takes two parameters of types T1 and T2 and returns a value of R  
18  * @tparam T1 the type of parameter 1  
19  * @tparam T2 the type of parameter 2  
20  * @tparam R the type of the result of function f  
21  * @return a value of R, wrapped in Try  
22  */  
23 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  
24
```

map3:

```

26 |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) // TO BE
27 |   * The map3 function. Much like map2
28 |   *
29 |   * @param t1y parameter 1 wrapped in Try
30 |   * @param t2y parameter 2 wrapped in Try
31 |   * @param t3y parameter 3 wrapped in Try
32 |   * @param f function that takes three parameters of types T1, T2 and T3 and returns a value of R
33 |   * @tparam T1 the type of parameter 1
34 |   * @tparam T2 the type of parameter 2
35 |   * @tparam T3 the type of parameter 3
36 |   * @tparam R the type of the result of function f
37 |   * @return a value of R, wrapped in Try
38 |   */
39 |
40 |

```

map7:

```
40
41 /**
42  * You get the idea...
43  */
44 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])
45   (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
46
```

lift:

```
47 /**
48  * Lift function to transform a function f of type T=>R into a function of type Try[T]>Try[R]
49  *
50  * @param f the function we start with, of type T=>R
51  * @tparam T the type of the parameter to f
52  * @tparam R the type of the result of f
53  * @return a function of type Try[T]>Try[R]
54  */
55 // You know this one
56 def lift[T, R](f: T => R): Try[T] => Try[R] = _ map f // TO BE IMPLEMENTED
57
```

lift2:

```
58 /**
59  * Lift function to transform a function f of type (T1,T2)=>R into a function of type (Try[T1],Try[T2])=>Try[R]
60  *
61  * @param f the function we start with, of type (T1,T2)=>R
62  * @tparam T1 the type of the first parameter to f
63  * @tparam T2 the type of the second parameter to f
64  * @tparam R the type of the result of f
65  * @return a function of type (Try[T1],Try[T2])=>Try[R]
66  */
67 // Think Simple, Elegant, Obvious
68 def lift2[T1, T2, R](f: (T1, T2) => R): (Try[T1], Try[T2]) => Try[R] = (t1y, t2y) => map2(t1y, t2y)(f) // TO BE IMPLEMENTED
69
```

lift3:

```
70 /**
71  * Lift function to transform a function f of type (T1,T2,T3)=>R into a function of type (Try[T1],Try[T2],Try[T3])=>Try[R]
72  *
73  * @param f the function we start with, of type (T1,T2,T3)=>R
74  * @tparam T1 the type of the first parameter to f
75  * @tparam T2 the type of the second parameter to f
76  * @tparam T3 the type of the third parameter to f
77  * @tparam R the type of the result of f
78  * @return a function of type (Try[T1],Try[T2],Try[T3])=>Try[R]
79  */
80 // If you can do lift2, you can do lift3
81 def lift3[T1, T2, T3, R](f: (T1, T2, T3) => R): (Try[T1], Try[T2], Try[T3]) => Try[R] = (t1y, t2y, t3y) => map3(t1y, t2y, t3y)(f) // TO BE IMPLEMENTED
82
```

lift7:

```
84 * Lift function to transform a function f of type (T1,T2,T3,T4,T5,T6,T7)=>R into a function of type (Try[T1],Try[T2],Try[T3],Try[T4],Try[T5],Try[T6],Try[T7])=>R
85 *
86 * @param f the function we start with, of type (T1,T2,T3,T4,T5,T6,T7)=>R
87 * @tparam T1 the type of the first parameter to f
88 * @tparam T2 the type of the second parameter to f
89 * @tparam T3 the type of the third parameter to f
90 * @tparam T4 the type of the fourth parameter to f
91 * @tparam T5 the type of the fifth parameter to f
92 * @tparam T6 the type of the sixth parameter to f
93 * @tparam T7 the type of the seventh parameter to f
94 * @tparam R the type of the result of f
95 * @return a function of type (Try[T1],Try[T2],Try[T3],Try[T4],Try[T5],Try[T6],Try[T7])=>Try[R]
96 */
97 // If you can do lift3, you can do lift7
98 def lift7[T1, T2, T3, T4, T5, T6, T7, R](f: (T1, T2, T3, T4, T5, T6, T7) => R):
99 (Try[T1], Try[T2], Try[T3], Try[T4], Try[T5], Try[T6], Try[T7]) => Try[R] = (t1y, t2y, t3y, t4y, t5y, t6y, t7y) => map7(t1y, t2y, t3y, t4y, t5y, t6y, t7y)(f) // TO BE
```

invert2:

```
101 /**
102  * This method inverts the order of the first two parameters of a two-(or more-)parameter curried function.
103  *
104  * @param f the function
105  * @tparam T1 the type of the first parameter
106  * @tparam T2 the type of the second parameter
107  * @tparam R the result type
108  * @return a curried function which takes the second parameter first
109  */
110 // Hint: think about writing an anonymous function that takes a t2, then a t1 and returns the appropriate result
111 // NOTE: you won't be able to use the "_" character here because the compiler infers an ordering that you don't want
112 def invert2[T1, T2, R](f: T1 => T2 => R): T2 => T1 => R = t2 => t1 => f(t1)(t2) // TO BE IMPLEMENTED
```

invert3:

```
114 /**
115  * This method inverts the order of the first three parameters of a three-(or more-)parameter curried function.
116  *
117  * @param f the function
118  * @tparam T1 the type of the first parameter
119  * @tparam T2 the type of the second parameter
120  * @tparam T3 the type of the third parameter
121  * @tparam R the result type
122  * @return a curried function which takes the third parameter first, then the second, etc.
123  */
124 // If you can do invert2, you can do this one too
125 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
```

invert4:

```
127 /**
128  * This method inverts the order of the first four parameters of a four-(or more-)parameter curried function.
129  *
130  * @param f the function
131  * @tparam T1 the type of the first parameter
132  * @tparam T2 the type of the second parameter
133  * @tparam T3 the type of the third parameter
134  * @tparam T4 the type of the fourth parameter
135  * @tparam R the result type
136  * @return a curried function which takes the fourth parameter first, then the third, etc.
137  */
138 // If you can do invert3, you can do this one too
139 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
140
```

uncurried2:

```
141 /**
142  * This method uncurries the first two parameters of a three- (or more-)
143  * parameter curried function.
144  * The result is a (curried) function whose first parameter is a tuple of the first two parameters of f;
145  * whose second parameter is the third parameter, etc.
146  *
147  * @param f the function
148  * @tparam T1 the type of the first parameter
149  * @tparam T2 the type of the second parameter
150  * @tparam T3 the type of the third parameter
151  * @tparam R the result type of function f
152  * @return a (curried) function of type (T1,T2)=>T3=>R
153  */
154 // This one is a bit harder. But again, think in terms of an anonymous function that is what you want to return
155 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
```

uncurried3:

```
157 /**
158  * This method uncurries the first three parameters of a four- (or more-)
159  * parameter curried function.
160  * The result is a (curried) function whose first parameter is a tuple of the first three parameters of f;
161  * whose second parameter is the third parameter, etc.
162  *
163  * @param f the function
164  * @tparam T1 the type of the first parameter
165  * @tparam T2 the type of the second parameter
166  * @tparam T3 the type of the third parameter
167  * @tparam T4 the type of the fourth parameter
168  * @tparam R the result type of function f
169  * @return a (curried) function of type (T1,T2,T3)=>T4=>R
170  */
171 // If you can do uncurried2, then you can do this one
172 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
```

uncurried7:

```
179 * @param f the function
180 * @tparam T1 the type of the first parameter
181 * @tparam T2 the type of the second parameter
182 * @tparam T3 the type of the third parameter
183 * @tparam T4 the type of the fourth parameter
184 * @tparam R the result type of function f
185 * @return a (curried) function of type (T1,T2,T3)=>T4=>R
186 */
187 // If you can do uncurried3, then you can do this one
188 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 =
189 (t1, t2, t3, t4, t5, t6, t7) => t8 => f(t1)(t2)(t3)(t4)(t5)(t6)(t7)(t8) // TO BE IMPLEMENTED
190
```

MoviesProtocol:

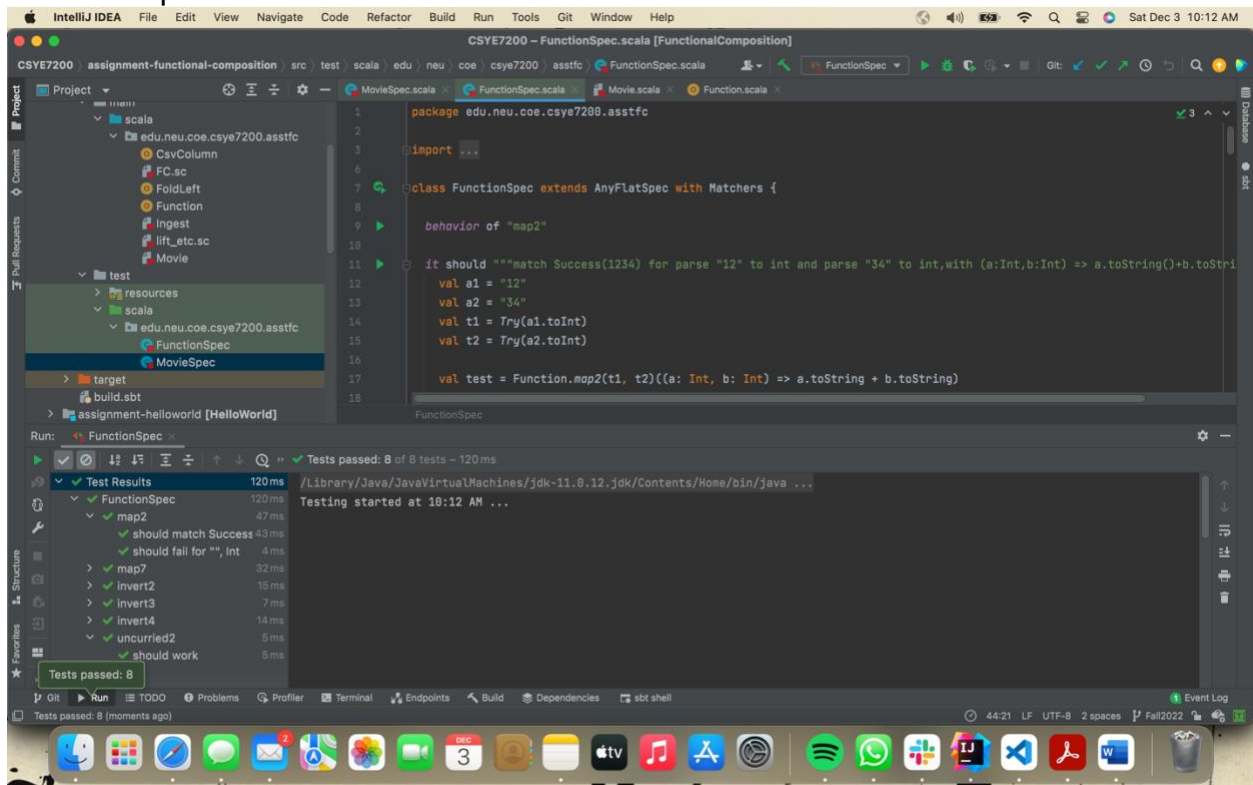
```
101 //Hint: You may refer to the slides discussed in class for how to serialize object to json
102 object MoviesProtocol extends DefaultJsonProtocol {
103   // 20 points
104   // TO BE IMPLEMENTED
105   implicit val formatFormat = jsonFormat4(Format.apply)
106   implicit val nameFormat = jsonFormat4(Name.apply)
107   implicit val ratingFormat = jsonFormat2(Rating.apply)
108   implicit val reviewFormat = jsonFormat7(Reviews.apply)
109   implicit val productionFormat = jsonFormat4(Production.apply)
110   implicit val principalFormat = jsonFormat2(Principal.apply)
111   implicit val movieJsonFormat = jsonFormat11(Movie.apply)
112 }
```

## testSerializationAndDeserialization:

```
128 //Hint: Serialize the input to Json format and deserialize back to Object, check the result is still equal to original input.
129 def testSerializationAndDeserialization(ms: Seq[Movie]): Boolean = {
130   // 5 points
131   // TO BE IMPLEMENTED
132   import MoviesProtocol._
133   val json = ms.toJson
134   val movies = json.convertTo[Seq[Movie]]
135   ms == movies
136 }
```

## -Unit tests

### FunctionSpec.scala:





## MovieSpec.scala:

