Hrishikesh Sanjay Pawar – SEC01 (NUID 002707307)

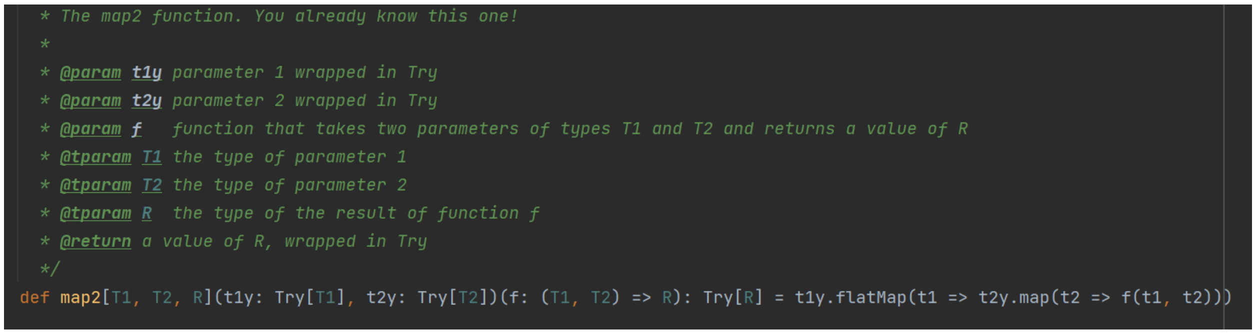
Big Data System Engineering with Scala
Spring 2023
Assignment No. 5

-List of Tasks Implemented

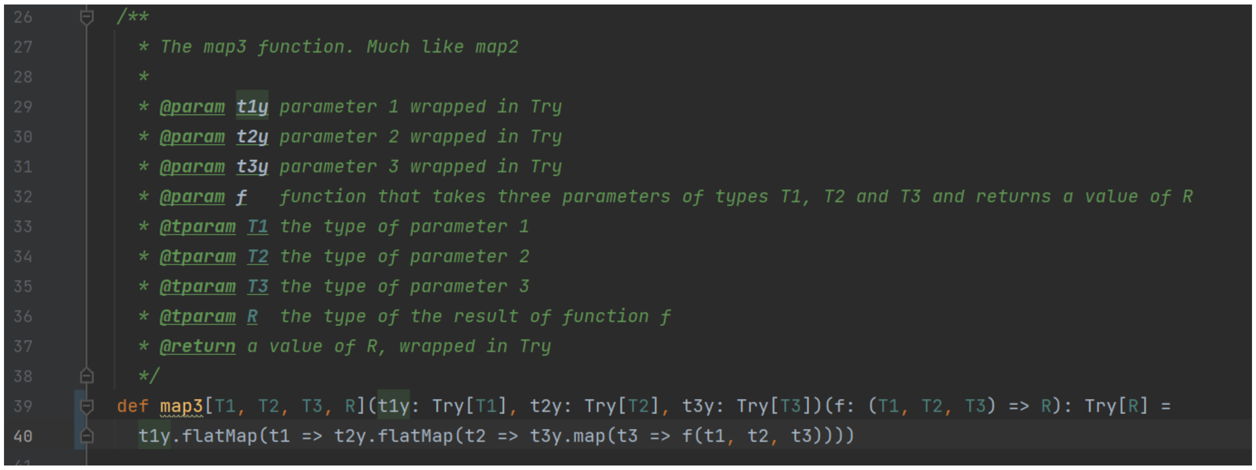
Implemented the incomplete parts in Movie.scala and Function.scala files.

-Code

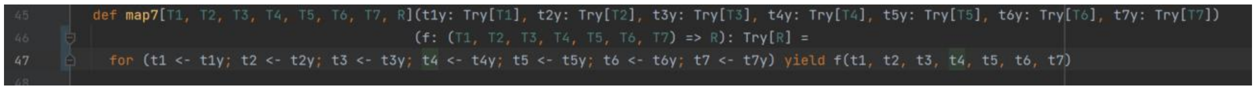
Function.scala



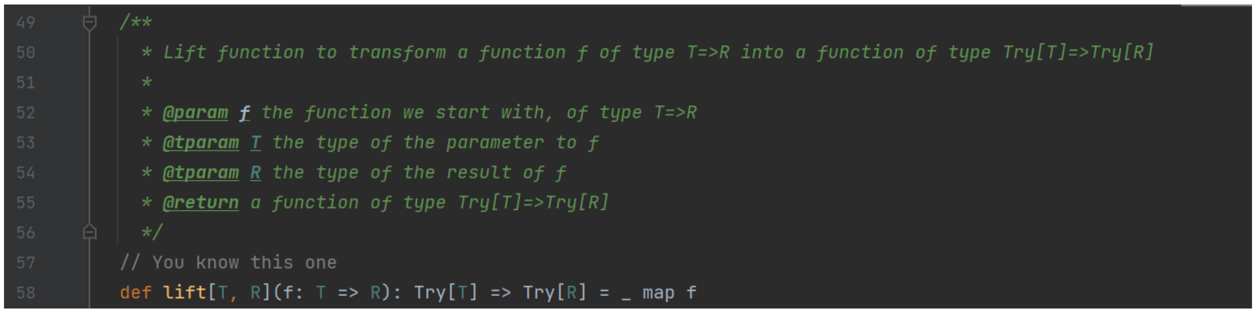
```scala
 * The map2 function. You already know this one!
 *
 * @param t1y parameter 1 wrapped in Try
 * @param t2y parameter 2 wrapped in Try
 * @param f   function that takes two parameters of types T1 and T2 and returns a value of R
 * @tparam T1 the type of parameter 1
 * @tparam T2 the type of parameter 2
 * @tparam R  the type of the result of function f
 * @return a value of R, wrapped in Try
 */
def map2[T1, T2, R](t1y: Try[T1], t2y: Try[T2])(f: (T1, T2) => R): Try[R] = t1y.flatMap(t1 => t2y.map(t2 => f(t1, t2)))
```



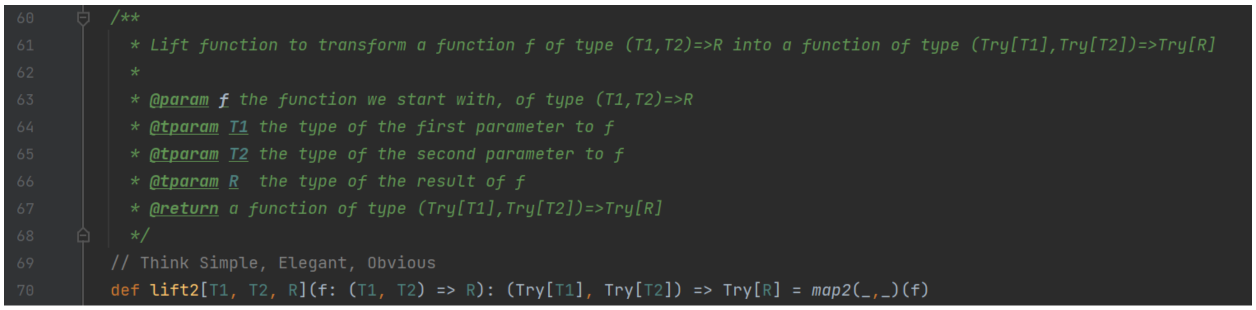
```scala
/**
  * The map3 function. Much like map2
  *
  * @param t1y parameter 1 wrapped in Try
  * @param t2y parameter 2 wrapped in Try
  * @param t3y parameter 3 wrapped in Try
  * @param f   function that takes three parameters of types T1, T2 and T3 and returns a value of R
  * @tparam T1 the type of parameter 1
  * @tparam T2 the type of parameter 2
  * @tparam T3 the type of parameter 3
  * @tparam R  the type of the result of function f
  * @return a value of R, wrapped in Try
  */
def map3[T1, T2, T3, R](t1y: Try[T1], t2y: Try[T2], t3y: Try[T3])(f: (T1, T2, T3) => R): Try[R] =
  t1y.flatMap(t1 => t2y.flatMap(t2 => t3y.map(t3 => f(t1, t2, t3))))
```



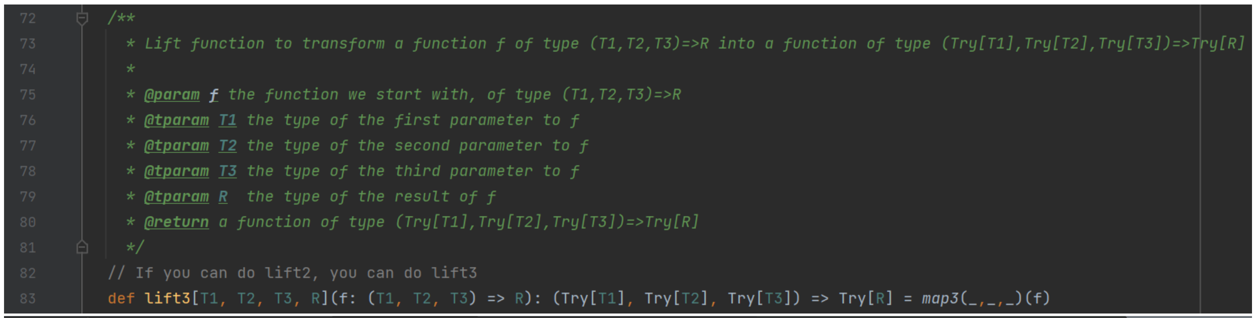
```scala
  def map7[T1, T2, T3, T4, T5, T6, T7, R](t1y: Try[T1], t2y: Try[T2], t3y: Try[T3], t4y: Try[T4], t5y: Try[T5], t6y: Try[T6], t7y: Try[T7])
                                         (f: (T1, T2, T3, T4, T5, T6, T7) => R): Try[R] =
    for (t1 <- t1y; t2 <- t2y; t3 <- t3y; t4 <- t4y; t5 <- t5y;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



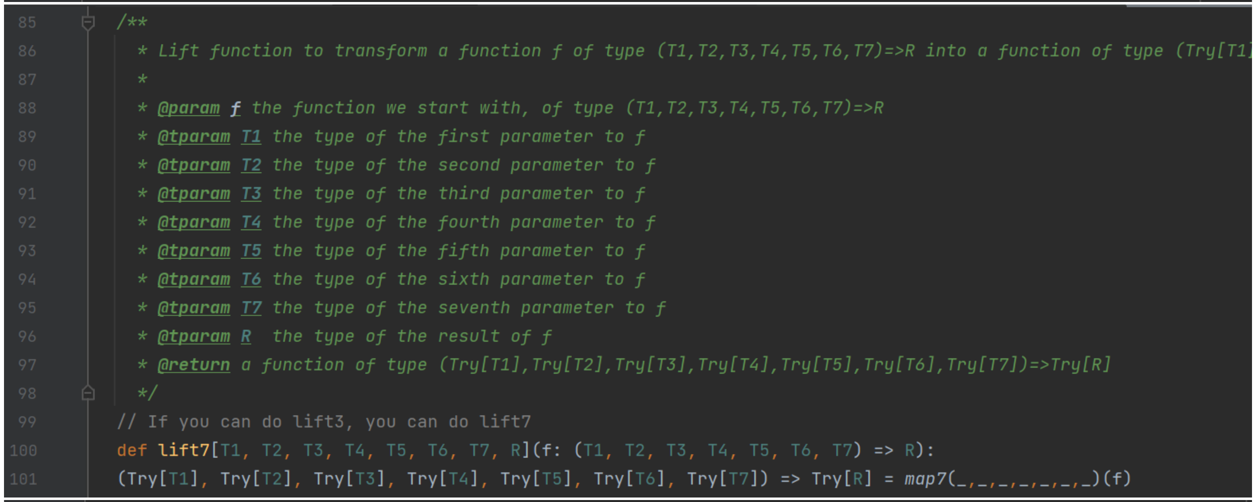
```scala
/**
  * Lift function to transform a function f of type T=>R into a function of type Try[T]=>Try[R]
  *
  * @param f the function we start with, of type T=>R
  * @tparam T the type of the parameter to f
  * @tparam R the type of the result of f
  * @return a function of type Try[T]=>Try[R]
  */
// You know this one
def lift[T, R](f: T => R): Try[T] => Try[R] = _ map f
```



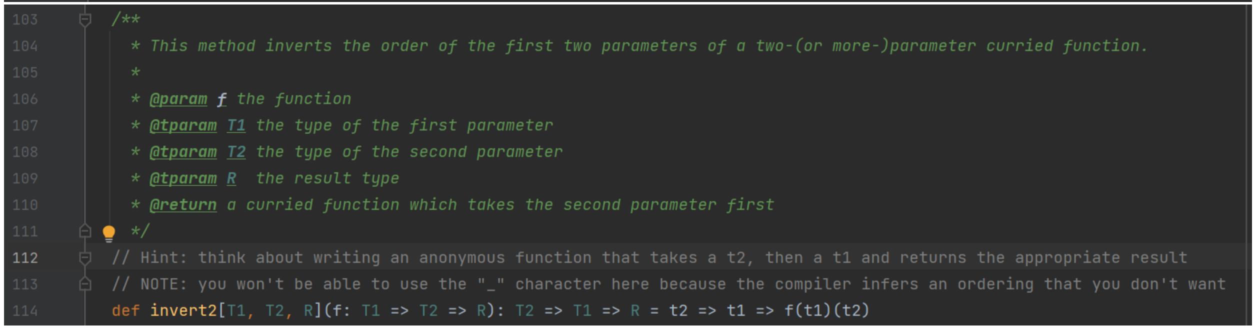
```scala
/**
  * Lift function to transform a function f of type (T1,T2)=>R into a function of type (Try[T1],Try[T2])=>Try[R]
  *
  * @param f the function we start with, of type (T1,T2)=>R
  * @tparam T1 the type of the first parameter to f
  * @tparam T2 the type of the second parameter to f
  * @tparam R  the type of the result of f
  * @return a function of type (Try[T1],Try[T2])=>Try[R]
  */
// Think Simple, Elegant, Obvious
def lift2[T1, T2, R](f: (T1, T2) => R): (Try[T1], Try[T2]) => Try[R] = map2(_,_)(f)
```



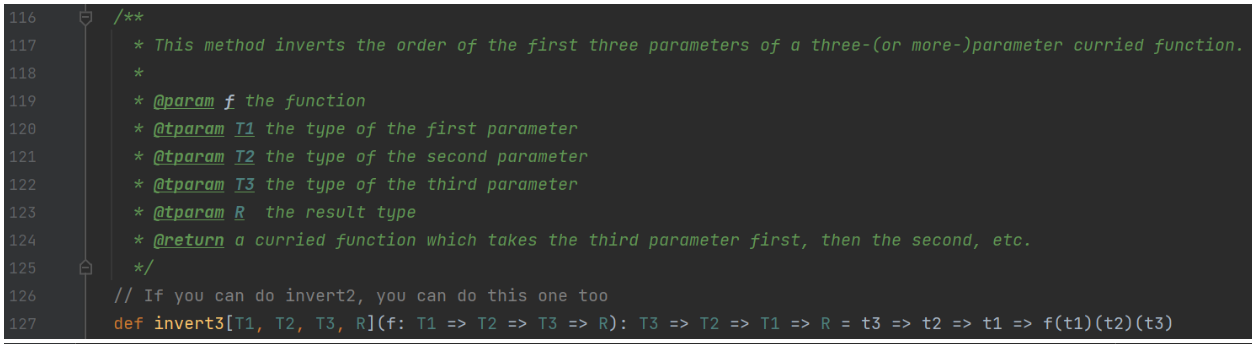
```scala
/**
  * Lift function to transform a function f of type (T1,T2,T3)=>R into a function of type (Try[T1],Try[T2],Try[T3])=>Try[R]
  *
  * @param f the function we start with, of type (T1,T2,T3)=>R
  * @tparam T1 the type of the first parameter to f
  * @tparam T2 the type of the second parameter to f
  * @tparam T3 the type of the third parameter to f
  * @tparam R  the type of the result of f
  * @return a function of type (Try[T1],Try[T2],Try[T3])=>Try[R]
  */
// If you can do lift2, you can do lift3
def lift3[T1, T2, T3, R](f: (T1, T2, T3) => R): (Try[T1], Try[T2], Try[T3]) => Try[R] = map3(_,_,_)(f)
```



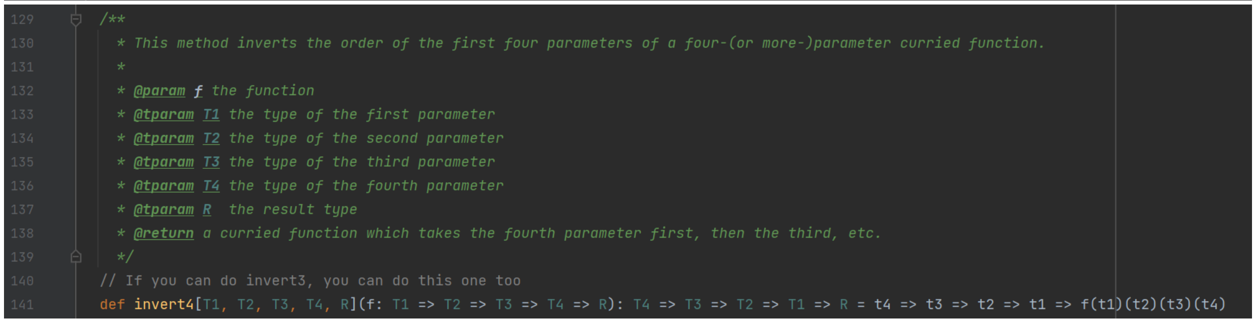
```scala
/**
  * Lift function to transform a function f of type (T1,T2,T3,T4,T5,T6,T7)=>R into a function of type (Try[T1
  *
  * @param f the function we start with, of type (T1,T2,T3,T4,T5,T6,T7)=>R
  * @tparam T1 the type of the first parameter to f
  * @tparam T2 the type of the second parameter to f
  * @tparam T3 the type of the third parameter to f
  * @tparam T4 the type of the fourth parameter to f
  * @tparam T5 the type of the fifth parameter to f
  * @tparam T6 the type of the sixth parameter to f
  * @tparam T7 the type of the seventh parameter to f
  * @tparam R  the type of the result of f
  * @return a function of type (Try[T1],Try[T2],Try[T3],Try[T4],Try[T5],Try[T6],Try[T7])=>Try[R]
  */
// If you can do lift3, you can do lift7
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)
```



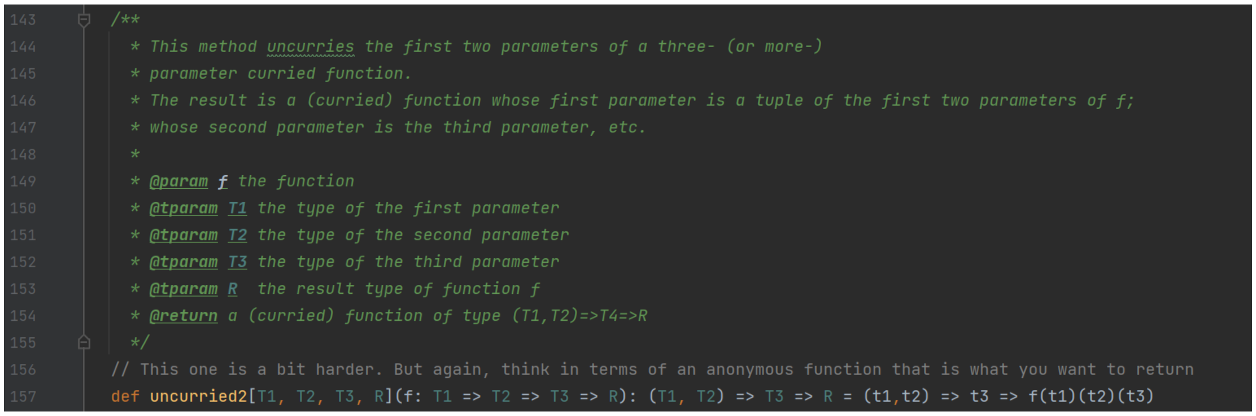
```scala
/**
  * This method inverts the order of the first two parameters of a two-(or more-)parameter curried function.
  *
  * @param f the function
  * @tparam T1 the type of the first parameter
  * @tparam T2 the type of the second parameter
  * @tparam R  the result type
  * @return a curried function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



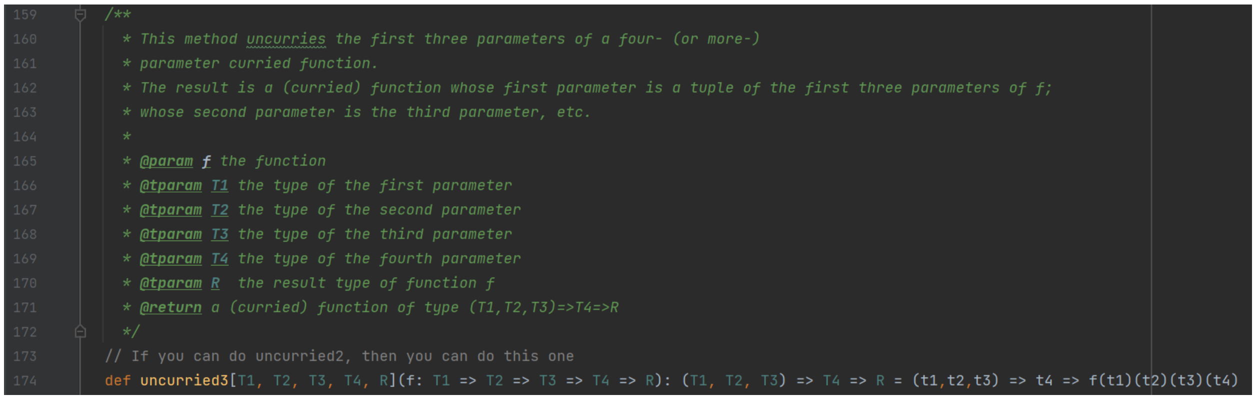
```scala
/**
 * This method inverts the order of the first three parameters of a three-(or more-)parameter curried function.
 *
 * @param f the function
 * @tparam T1 the type of the first parameter
 * @tparam T2 the type of the second parameter
 * @tparam T3 the type of the third parameter
 * @tparam R  the result type
 * @return a curried function which takes the third parameter first, then the second, etc.
 */
// If you can do invert2, you can do this one too
def invert3[T1, T2, T3, R](f: T1 => T2 => T3 => R): T3 => T2 => T1 => R = t3 => t2 => t1 => f(t1)(t2)(t3)
```



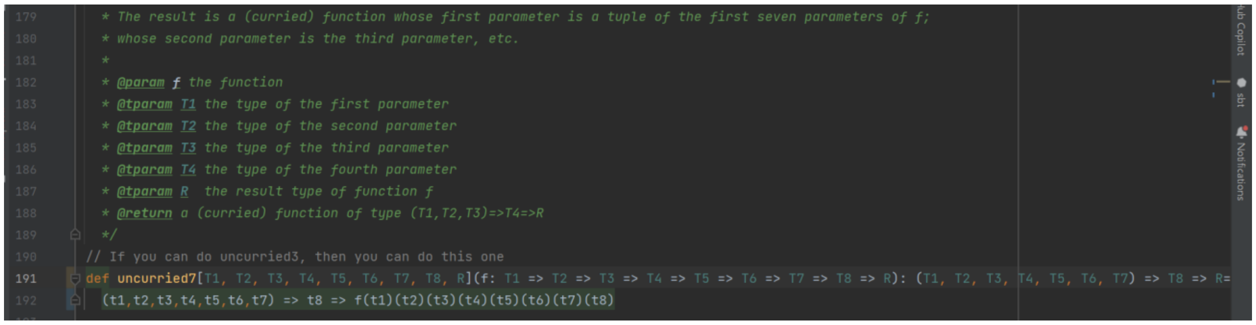
```scala
/**
 * This method inverts the order of the first four parameters of a four-(or more-)parameter curried function.
 *
 * @param f the function
 * @tparam T1 the type of the first parameter
 * @tparam T2 the type of the second parameter
 * @tparam T3 the type of the third parameter
 * @tparam T4 the type of the fourth parameter
 * @tparam R  the result type
 * @return a curried function which takes the fourth parameter first, then the third, etc.
 */
// If you can do invert3, you can do this one too
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)
```



```scala
/**
 * This method uncurries the first two parameters of a three- (or more-)
 * parameter curried function.
 * The result is a (curried) function whose first parameter is a tuple of the first two parameters of f;
 * whose second parameter is the third parameter, etc.
 *
 * @param f the function
 * @tparam T1 the type of the first parameter
 * @tparam T2 the type of the second parameter
 * @tparam T3 the type of the third parameter
 * @tparam R  the result type of function f
 * @return a (curried) function of type (T1,T2)=>T4=>R
 */
// This one is a bit harder. But again, think in terms of an anonymous function that is what you want to return
def uncurried2[T1, T2, T3, R](f: T1 => T2 => T3 => R): (T1, T2) => T3 => R = (t1,t2) => t3 => f(t1)(t2)(t3)
```

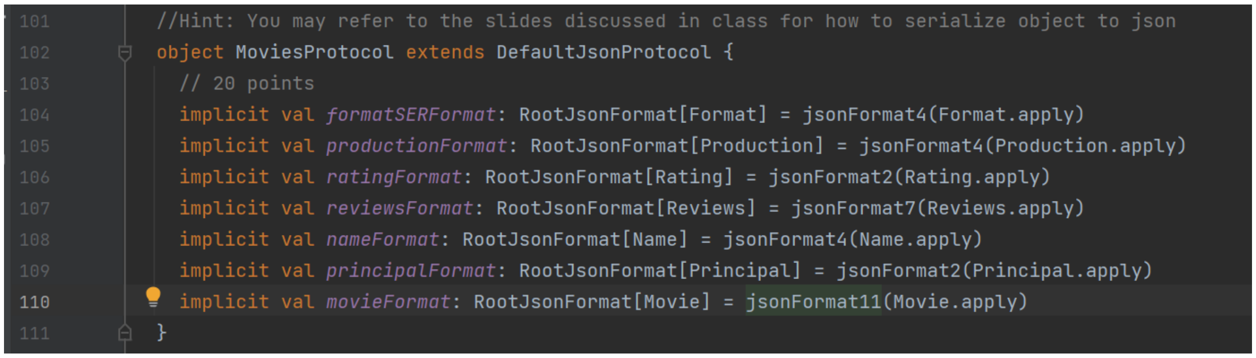


```scala
/**
 * This method uncurries the first three parameters of a four- (or more-)
 * parameter curried function.
 * The result is a (curried) function whose first parameter is a tuple of the first three parameters of f;
 * whose second parameter is the third parameter, etc.
 *
 * @param f the function
 * @tparam T1 the type of the first parameter
 * @tparam T2 the type of the second parameter
 * @tparam T3 the type of the third parameter
 * @tparam T4 the type of the fourth parameter
 * @tparam R  the result type of function f
 * @return a (curried) function of type (T1,T2,T3)=>T4=>R
 */
// If you can do uncurried2, then you can do this one
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)
```

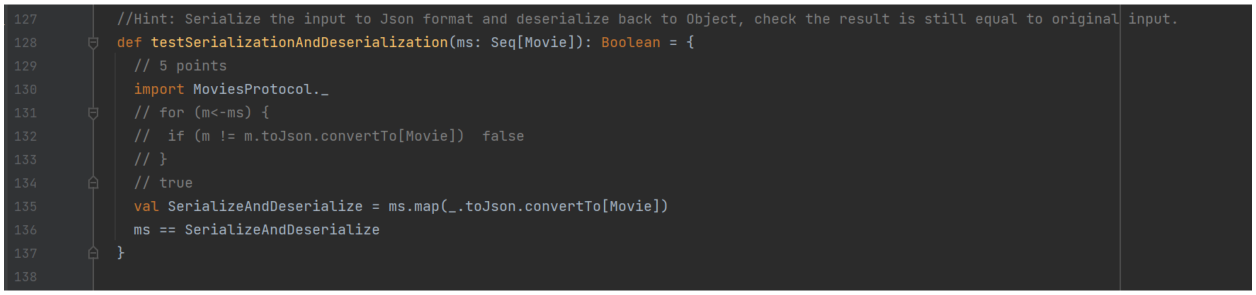


```scala
   * The result is a (curried) function whose first parameter is a tuple of the first seven parameters of f;
   * whose second parameter is the third parameter, etc.
   *
   * @param f the function
   * @tparam T1 the type of the first parameter
   * @tparam T2 the type of the second parameter
   * @tparam T3 the type of the third parameter
   * @tparam T4 the type of the fourth parameter
   * @tparam R  the result type of function f
   * @return a (curried) function of type (T1,T2,T3)=>T4=>R
   */
  // If you can do uncurried3, then you can do this one
  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)
```

Movie.scala



```scala
  //Hint: You may refer to the slides discussed in class for how to serialize object to json
  object MoviesProtocol extends DefaultJsonProtocol {
    // 20 points
    implicit val formatSERFormat: RootJsonFormat[Format] = jsonFormat4(Format.apply)
    implicit val productionFormat: RootJsonFormat[Production] = jsonFormat4(Production.apply)
    implicit val ratingFormat: RootJsonFormat[Rating] = jsonFormat2(Rating.apply)
    implicit val reviewsFormat: RootJsonFormat[Reviews] = jsonFormat7(Reviews.apply)
    implicit val nameFormat: RootJsonFormat[Name] = jsonFormat4(Name.apply)
    implicit val principalFormat: RootJsonFormat[Principal] = jsonFormat2(Principal.apply)
    implicit val movieFormat: RootJsonFormat[Movie] = jsonFormat11(Movie.apply)
  }
```

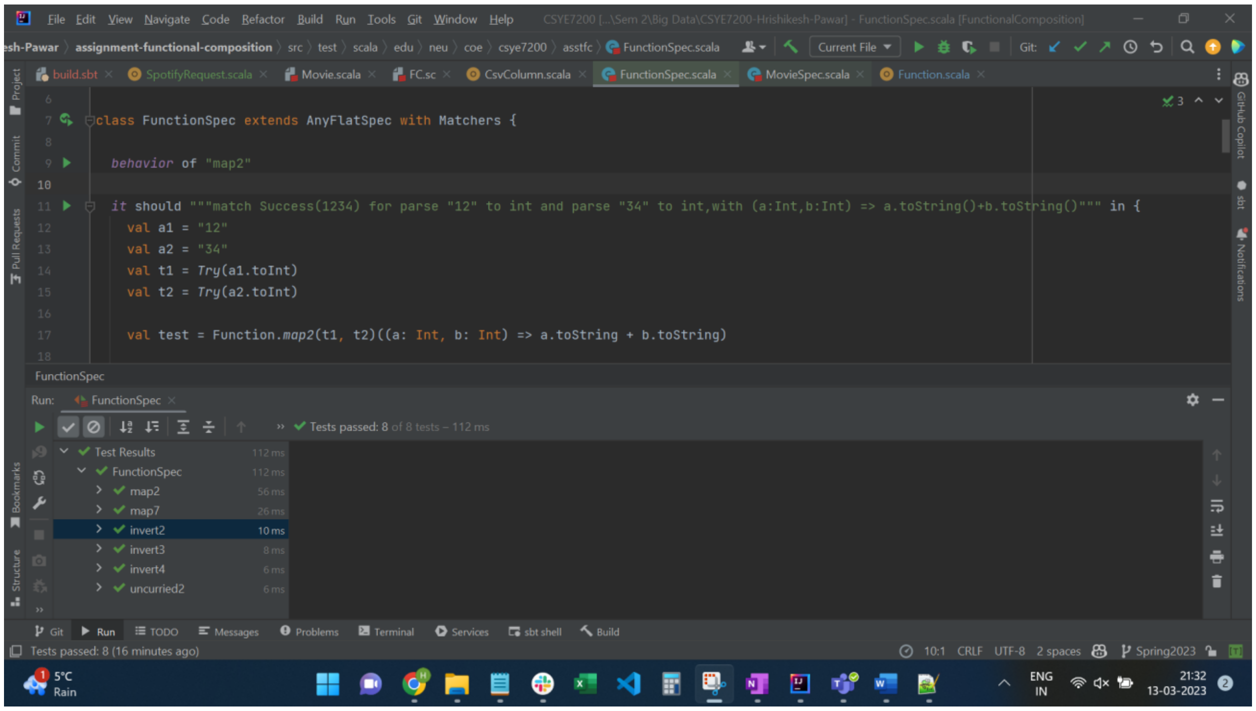


```scala
  //Hint: Serialize the input to Json format and deserialize back to Object, check the result is still equal to original input.
  def testSerializationAndDeserialization(ms: Seq[Movie]): Boolean = {
    // 5 points
    import MoviesProtocol._
    // for (m<-ms) {
    //  if (m != m.toJson.convertTo[Movie])  false
    // }
    // true
    val SerializeAndDeserialize = ms.map(_.toJson.convertTo[Movie])
    ms == SerializeAndDeserialize
  }
```

-Unit tests

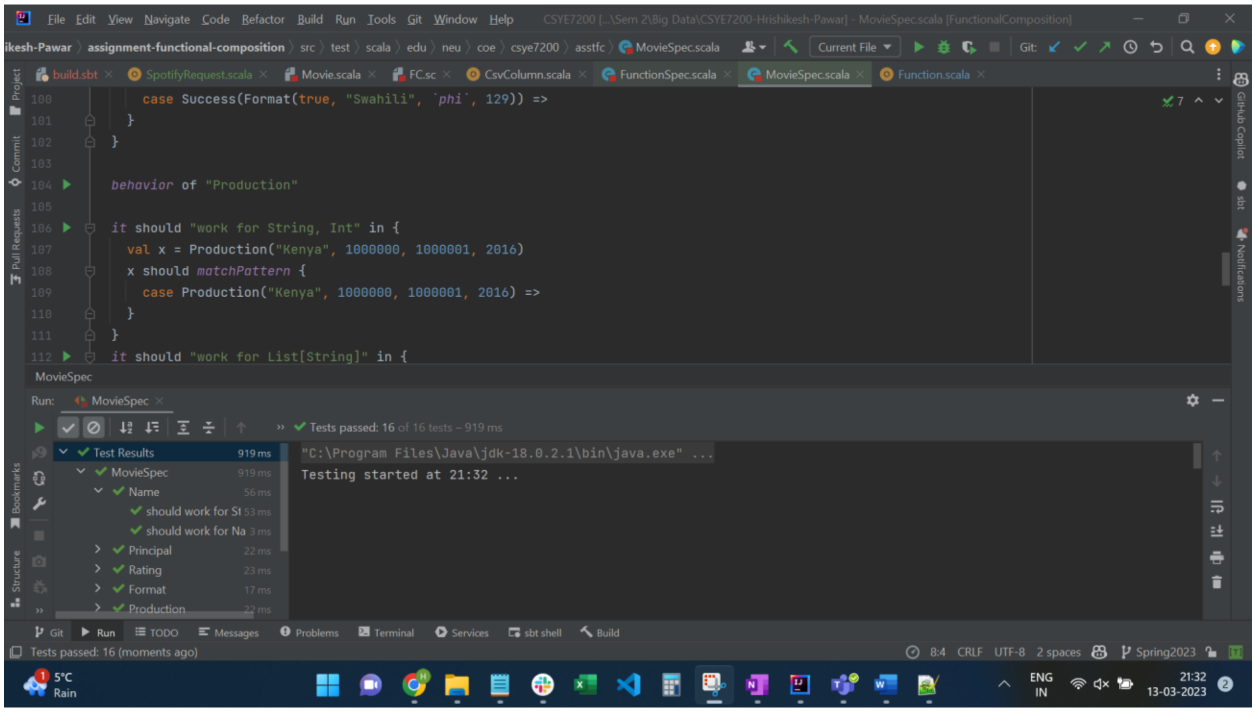
Paste Screenshots of your unit tests. (Make sure to include the time on your screen)

Incase Screenshot is not clear, paste a screenshot of all unit tests separately and a screenshot of the entire screen with the time included.



```scala
class FunctionSpec extends AnyFlatSpec with Matchers {

  behavior of "map2"

  it should """match Success(1234) for parse "12" to int and parse "34" to int,with (a:Int,b:Int) => a.toString()+b.toString()""" in {
    val a1 = "12"
    val a2 = "34"
    val t1 = Try(a1.toInt)
    val t2 = Try(a2.toInt)

    val test = Function.map2(t1, t2)((a: Int, b: Int) => a.toString + b.toString)

```

```
Tests passed: 8 of 8 tests – 112 ms

Test Results     112 ms
  FunctionSpec   112 ms
    map2          56 ms
    map7          26 ms
    invert2       10 ms
    invert3        8 ms
    invert4        6 ms
    uncurried2     6 ms

Tests passed: 8 (16 minutes ago)
```



```scala
        case Success(Format(true, "Swahili", `phi`, 129)) =>
      }
    }

  behavior of "Production"

  it should "work for String, Int" in {
    val x = Production("Kenya", 1000000, 1000001, 2016)
    x should matchPattern {
      case Production("Kenya", 1000000, 1000001, 2016) =>
    }
  }
  it should "work for List[String]" in {
```

```
Tests passed: 16 of 16 tests – 919 ms

Test Results                919 ms
  MovieSpec                 919 ms
    Name                     56 ms
      should work for St     53 ms
      should work for Na      3 ms
    Principal                22 ms
    Rating                   23 ms
    Format                   17 ms
    Production               22 ms

"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
Testing started at 21:32 ...

Tests passed: 16 (moments ago)
```

-GitHub Link

https://github.com/hrishikesh-pawar/CSYE7200-Hrishikesh-Pawar/tree/Spring2023/assignment-functional-composition