Spark-Scala

1. Write a lambda function in Scala, using map operation, which takes a sequence of salaries as input and outputs double of every element from input.

A lambda function in Scala (as in many other languages as python or Java) is a function used in the functional programming paradigm, that it is defined without a name to identify it. In my opinion, it uses a mathematical definition, more similar as the functions are defined in maths. So it has got a lot of spread and most languages, although they are not functional ones, have include lambda functions (Java is a OOP but it has included lambda functions in Java version 8).

The enunciate said that “using a lambda function” so being strict, the answer is “map(valor => valor*2). But this answer has not context. So I have wrote some code snippets to be able to give a meaning to the lambda function is ask.

For instance, using a class Salaries, with a method for doubling the salaries are coming as input:

```
class Salaries(val sal:Seq[Double]) {

  def doubleThoseSalaries():Seq[Double] = {
    sal.map(salario => salario*2)
  }

}

object Ejercicio {
  def main(args:Array[String]) {

    val seqSalaries = Seq(1000.00, 1500.90)
    val output = new Salaries(seqSalaries).doubleThoseSalaries()
    output.foreach(println)
  }
}
```

You could use a object approach instead:

```
object DoubleSalaries {

  def increaseSalaries(salaries:Seq[Double]):Seq[Double] = {
    salaries.map(salario => salario *2)

  }

  def main(args: Array[String]):Unit ={
   //suponemos que el input es una estructura de datos de tipo Seq
    val salaries = Seq(1001.00, 2756.33)

    val output = increaseSalaries(salaries)

    output.foreach(println)

  }
}
```

As you can see, there are several possibilities.

2. What are the types of program evaluation in Scala?

Scala has 2 different types of program evaluation

1. Strict Evaluation
2. Lazy Evaluation

The evaluation can be in values or in arguments.

Note: The question is done in a general fashion. A particular case of evaluation is when the arguments are used by a function. Then, we use the terminology "call-by-value (strict evaluation) and "call-by-name" (lazy evaluation). As the question is general, and as the question 6th is asking about the parameter evalutation, I have included examples only in the question 6th as it is a particular approach, so better for doing an example.

In an Strict evaluation (also called Eager), any expression is evaluated as soon as it is bound to a variable. This means that for instance, an error for assigning not allowed values are raised at compilation time.
The opposite is the Lazy evaluation, when the expression is delayed to the moment when the value is needed. This means that error will be raised at execution time.

Note: check above the answer of question 6 for getting an example when calling-by-name versus calling-by-value. A strict function in Scala takes its arguments by value rather than by name.

3. Can a companion object in Scala access the private members of its companion class in Scala?

Yes, they can. It is an exception, as "private" by definition means that only can be accessed within the class. But companion objects are created to make the things easier to programmers :)

[METER UN EJEMPLO DE CLASS Y UN JAVAP, Y DESPUÉS SU COMPANION OBJECT Y HACER OTRO JAVAP PARA VER LA DIFERENCIA]

4. What is pattern matching?

It is checking a value against a pattern. In Scala, you can use the "match" operator alongside "case", and use a regex to evaluate the value with "match"

For instance

The initial requirement is to check if the value of the variable "myVariable" is equal or not to 33:

```
scala> val miVariable = 33
miVariable: Int = 33

scala> val resultado = miVariable match {
     |   case 33 => "acierto"
     |   case _ => "fallo"
     | }
resultado: String = acierto

scala> println(resultado)
acierto
```

But if it is required to get a more general case, for instance, “any number starting by 3”, you must use regular expressions (regex). A regex is used to identify if a value is according to a given pattern.

If the case is “any number starting by 3”, you can rewrite the previous code to be able to identify any value matching the requirement

Let’s navigate to a website as https://regex101.com/ (it is my preferred option), to define a pattern meaning “any number starting by 3”. Bear in mind that the regex are applied to Strings, so we are going to cast the value of “myVariable” to String before the match

The regex for “any string starting by 3” is ^3+ (^ stands for checking at the begining of the string, 3 is the value to check and the “+” stands for “one or many times”, so we must import the regex classes in the scala REPL, defining the pattern as a regex, and check with the “match/case” as we did before.

```
scala> import scala.util.matching._
import scala.util.matching._

scala> val patron = """^3+""".r                          //patron a utilizar
patron: scala.util.matching.Regex = ^3+

scala> myVariable.toString match {
     | case patron(_*) => "has acertado"
     | case _ => "has fallado"
     | }
res2: String = has acertado
```

5. What is an anonymous object in Scala?

It is a object that it is create (“new Object”) but it is not assigned to any identifier, so the object cannot be used more than once.

For instance

```
scala> class Orange {
     |   def show() = {
     |     println("Hello")
     |   }
     | }
defined class Orange

scala> object Prueba {
     | def main(args:Array[String]) = {
     |   new Orange().show()       //creamos el objeto sin asignarlo a un identificador
     | }
     | }
defined object Prueba

scala> Prueba.main(null)
Hello
```

6. What is the difference between call-by-value and call-by-name function parameters?ç

Scala supports both. The difference is when the parameters are evaluated when calling the function

a) call-by-name: the parameters are evaluated when they are used, rather than the function is called.

b) call-by-value: the parameters are evaluated when the function is called and it is not more evaluated

In Scala, we use the type by using call-by-value (for instance “Int”) and a “proxy to the type” by using call-by-name (“=> Int”), the former is similar to a function that it takes no parameters and returns an “Int”.

For instance

We have a function that use twice the passed parameter. For instance

def miFuncion(x:Int, y:Int) = x*x

So when the function is using the parameter x, we can have a different number of steps

If miFuncion(2+3,5) was called call-by-name, the 2+3 was calculated twice, so, if we imagine the steps when the program is executed, it means (2+3) * (2+3), so 5 * (2+3) so 5 * 5 and then the result is 25.
If miFuncion(2+3,5) was called call-by-value, the 2+3 is calculated once, so the value is cached and used by the function as many times as it is needed but it is not calculated nevermore, so it means that (2+3) is calculated before calling the function, (2+3)=5, so the value is used inside the function 5*5 and the result is 25. The result is achieved in lesser steps.

For instance,

```
scala> class byName {
     | def suma(x: => Int, y:Int) = {
     | x+x
     | }
     | }
defined class byName

scala> val name = new byName().suma(2+3,5)
name: Int = 10

scala> class byValue {
     | def suma(x: Int, y:Int) = {
     | x+x
     | }
     | }
defined class byValue

scala> val value = new byValue().suma(2+3,5)
value: Int = 10
```

The result is the same (“10”) in both cases, but the steps are different. In byName, the 2+3 is calculated twice, because x is called twice. In byValue, the 2+3 is calculated, cached, and then used as many times as the function needs.

For showing this, we are going to make a small change in the calling function, to include a println.

```
scala> val name = new byName().suma({println("me han evaluado");2+3},5)
me han evaluado
me han evaluado
name: Int = 10

scala> val value = new byValue().suma({println("me han evaluado");2+3},5)
```

```
me han evaluado
value: Int = 10
```

You can see that as “x” is readed twice, when calling by name, the println is executed twice.

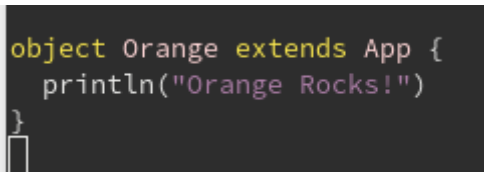
7. What is the use of Scala’s App?

App is a trait to be easier executing the code.

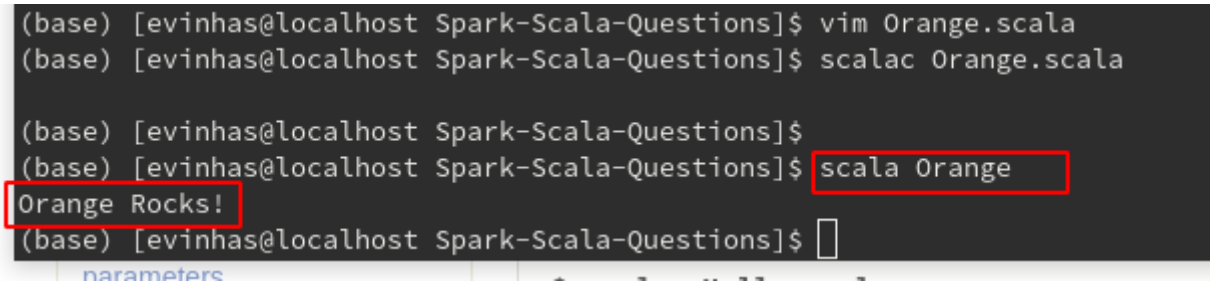
If you want to launch an application, there are 2 ways
- as in Java, define a “main” method (in Scala, we use “Objects” for doing this)
- extending the App trait

So extending App, we can run the cod in the body of the object without using a “main” method.

Example



So if we put this bunch of code in a file called “Orange.scala”, and we compile it with scalac, we can call the object although it doesn’t have a proper main method



8. How many types of constructors are used in Scala?

There are 2 types
- primary
- auxiliary

This is different than in Java.

The auxiliary constructor directly or indirectly invokes the primary constructor.

Example

```
scala> class Eduardo () {
     |   var contratar:Boolean = true
     |   def contratoAEduardo()={
     |     println("Eduardo podría ser una gran incorporacion  para Orange, por lo tanto,
¿lo contrato? " + contratar)
     |   }
```

```
     |   def this(contratar:Boolean){              //definimos un constructor auxiliar
     |     this()     //usamos el constructor primario
     |     this.contratar=true    //independiente del valor con el que se crea el objeto,
contratar va a ser true
     |   }
     | }
defined class Eduardo

scala> new Eduardo().contratoAEduardo()
Eduardo podría ser una gran incorporacion  para Orange, por lo tanto, ¿lo contrato? True

scala> new Eduardo(false).contratoAEduardo()   //aunque quiera pasar false a contratar no
puedo
Eduardo podría ser una gran incorporacion  para Orange, por lo tanto, ¿lo contrato? true
```

9. What is higher order function in Scala?

The possibility of passing a function as argument to another function or as return value from a function.

It is very useful, and it is posible because in Scala, the functions are first-class values.

The typical example is using the “println” function as argument for using “foreach”, and print on the screen all elements of a Scala collection.

Example:

```
scala> val listaPruebas = List(1,2,3,4)
listaPruebas: List[Int] = List(1, 2, 3, 4)

scala> listaPruebas.take(4).foreach(println)
1
2
3
4
```

In the previous example, the function “println” is the argument being passed to the “foreach” function, so for each element inside the loop, the function “println” is being applied

10. What is Scala Trait?

It is similar to an interface in Java. The Trait are used to being able to define how the object is used, defining the methods to deal with the objects, but the implementation is not defined.

So when a programmer defines a trait, it is defining how you can extend the code, in a controlled manner.

Example

```
trait telefonoMovil {
  def llamar
  def chatear
 }
```

Un movil puede ser un trait generico y ahora ir extendiendolo con marcas concretas

```
class XiaomiMi9 extends telefonoMovil{
  def llamar = {
```

```
    println("coger el telefono y marcar un numero en el teclado tactil"
  (...)
}

class LadrillazoNokia extends telefonoMovil{
  def llamar = {
    println("coger el telefono y marcar un numero en el teclado fisico"
  }
  (....)
}
```

It allows to maintain easier the code (because you can change the implementation of the method and the rest of the program will works) or incluse allows other people to extend the code.

Finally, comment that a trait could extend another trait or if not define the methods, declare an abstract class.

11. What is a tail-recursive function in Scala?

It is a function that the last step is call to itself.

It is used in the functional programming paradign, not only in Scala.

It is useful when you have to calculate a recursive function, as factorial of a number. As you know, the factorial is the multiplication of any non-negative integer smaller than the integer you want the calculate the factorial, including the integer itself.

For instance, the factorial of "3" is 1*2*3 = 6.

So Scala compiler is optimized for dealing with such functions using a tail-revursive function.

Example,

```
scala> class demo {
     | def factorial (n: Int): BigInt = {
     | if (n == 0) 1 else factorial(n-1) * n
     | }
     | }
defined class demo

scala> new demo().factorial(3)
res13: BigInt = 6
```