01: ♥ Q1 – Q6 Scala Beginner Functional Programming interview Q&As

Posted on July 26, 2016 by Arulkumaran Kumaraswamipillai

6 tips to transforming your thinking from OOP/imperative programming to **functional programming** (i.e. FP)

Q1. What is a function in Scala?
A1. A function is a group of statements that performs a task. A Scala function declaration is of the form:

```

def functionName ([list of parameters]) : [return
   function body
   return [expr]
}
```

```
6
```

Scala simple function

“addOne” is a function that takes an “Int” and returns a result of type “Int”

```

object FunctionsInScala3 extends App {

  // a function that takes an argument of Int ty
  // and returns a result of Int type
  def addOne(x: Int): Int = {
      x + 1 //returns an int
  }

  //calls the above function
  val result = addOne(5); //returns 6
}

```

In FP “no statements, only expressions“

In functional languages, there are no statements, only expressions. The above code can be rewritten as following using “Int => Int” **lambda expression**. Takes an “Int” as arg and the body (i.e. x+1) returns an “Int ” as a result.

```

object FunctionalScala3 extends App {

  //Expression assigned to "addOne"
  val addOne = (x: Int) => x + 1

  //calls the above function
  val result = addOne(5); //returns 6
}

```

Anonymous(i.e. **NO NAME**) lambda expression can be assigned to a variable “addOne”. Expression can also be written as:

```

val addOne = (x: Int) => {x + 1}

```

16 Technical Key Areas

80+ step by step Java Tutorials

OR as

```

//Expression assigned to "addOne"
val addOne:Int => Int = (x: Int) => {x + 1}

```

So, defining your functions without giving them a name, and sprinkling your code with these types of "**function literals**" is similar to writing integer literals like 32.

In FP, functions can not only be assigned to variables, but also can be **passed around like objects**.

Functions are Objects, and Objects are Functions

Q2. Can you explain the statement that "a function" is an object in Scala?

A2. In Scala, everything is an Object. No primitives, no wrapper types, no auto boxing & unboxing, etc. Since, Scala is an Object Oriented Programming (**OOP**) & a functional programming (**FP**) language, "Functions are Objects", and also "Objects are Functions" as every value is an Object in Scala.

```

object FunctionsInScala extends App {

  def addOne(x: Int): Int = {
      x + 1 //returns an int
  }

  def applyAndAddOne(f: Function1[Int, Int], num
      f(number) + 1
  }

  //calls the above function
  val result = addOne(5); //returns 6
  println("result = " + result)


  val result2 = applyAndAddOne(addOne, 5)
  println("result2 = " + result2)

}

```

100+ Java pre-interview coding tests

How good are your .....?

Output:

```

result = 6
result2 = 7

```

The function “applyAndAddOne” takes a function as an argument. Since a function is an object, it can be passed around as an argument to other functions. It takes two arguments:

1) A function of type Functio1[Int, Int], which takes an Int as arg and returns an Int type.

2) A “number” of type “Int”.

We are passing the “addOne” function to “applyAndAddOne” function along with a value of 5. So, it invokes the “addOne” function first to add 1 to 5, and then adds another one to return a result of 7.

Functions that take other functions as parameters, or whose result is a function is known as **higher-order functions**.

Q3. In Scala, why do you have traits like **Function1**, **Function2**, **Function3**, etc up to **Function22**?
A3. A “trait” in Scala is like an interface in Java. A “trait” has more ways to be applied than an interface in Java. In the above example we used “Function1[Int, Int]” to take a single argument of type “Int” and return a type of Int.

Q. How will you declare a function that takes 3 arguments of type Int, and returns a result of type Boolean?
A. Function3[Int, Int, Int, Boolean]

When you declare something like the following,

```

(arg1:Int, arg2:Int, arg3:Int) => Boolean

```

The Scala compiler implicitly creates an implementation of the right function trait. E.g. **Function3**[Int, Int, Int, Boolean]

Q. How will you declare a function that takes 2 arguments of type Int, 2 arguments of type String and returns a result of type Boolean?
A. Function4[Int, Int, String, String, Boolean].

Q4. Can you explain the following code, and what will be the output?

```

package com.mytutorial

object FunctionsInScala2 extends App {

   val listOfNumbers:List[Int] = (1 to 10).toLis

   applyFilter( x => x % 2 != 0, listOfNumbers,
   applyFilter( x => x % 2 == 0, listOfNumbers,
   applyFilter( x => x % 5 == 0, listOfNumbers,

   def applyFilter(f: Int => Boolean, input:List
       val result = input.filter(f);
       println(comment + " = " + result);
   }
}

```

A4. The output will be

```

odd numbers = List(1, 3, 5, 7, 9)
even numbers = List(2, 4, 6, 8, 10)
multiples of 5 = List(5, 10)

```

1) "x => x % 2 != 0" is a function of the form "Int => Boolean". The Scala compiler will implicitly create an object implementation of the trait "**Function1[Int, Boolean]**". This is known as an "**Anonymous Function** ".

2) "**applyFilter**" is a function that takes three input parameters **1)** a function, **2)** a list, **3)** a string and returns nothing.

3) "val result = input.filter(f);" filters the list based on the predicate supplied as a function. If the function returns "true", that element is added to the result. "x % 2 != 0" returns true for the odd numbers, hence odd numbers are added to the list and so on for the even numbers, and multiples of 5.

Q5. Is there anything not quite right with the following Scala code?

```

package com.mytutorial

object FunctionalScala extends App {

  def addNumbers(input:List[Int]): Int = {
    var sum = 0;
    input.foreach(x => sum += x)
    sum;
  }

  def addOddNumbers(input:List[Int]): Int = {
    var sum = 0;
    input.foreach(x => if(x % 2 != 0)sum += x)
    return sum;
  }

  def addGt3Numbers(input:List[Int]): Int = {
    var sum = 0;
    input.foreach(x => if(x > 3) sum += x)
    sum;
  }

  println(addNumbers((1 to 6).toList))
  println(addOddNumbers((1 to 6).toList))
  println(addGt3Numbers((1 to 6).toList))

}

```

A5. Lacks code reuse. The condition as to what numbers to be added is known as a "**predicate**". This predicate can be added as a function and passed to single "**addNumbers**" function as shown below.

```

package com.mytutorial

object FunctionalScala extends App {

  def addNumbers(input:List[Int], predicate: Int
    var sum = 0;
    input.foreach(x => if (predicate(x)) sum +=
    sum;
  }

  println(addNumbers((1 to 6).toList, x => true)
  println(addNumbers((1 to 6).toList, x => x % 2
  println(addNumbers((1 to 6).toList, x => x > 3

}

```

Q6. In the “AddGivenNumberExample.scala” below

```

object AddGivenNumberExample extends App {
    val three = new AddGivenNumber(3);
    println(three.add(5)) //8
}

class AddGivenNumber (initialNumber: Int) {
    def add(numberToAdd: Int) = initialNumber  +
}

```

How can you modify the code in “AddGivenNumber” so that in “AddGivenNumberExample” object instead of “three.add(5)”, you can just do “**three(5)**“.

A6. Change the name of “add(….)” function in the class “AddGivenNumber” to “**apply(….)**” as shown below. In mathematics and computer science, **Apply** is a function that applies functions to arguments. “apply” is a method which get’s called on Function application

So, you can use it in a Scala object as “three.apply(5)” or as **three(5)** as shown below.

```

object AddGivenNumberExample extends App {
    val three = new AddGivenNumber(3);
    println(three(5)) // can also do three.apply
}

class AddGivenNumber (initialNumber: Int) {
    def apply(numberToAdd: Int) = initialNumber
}


```

So, this is same as instead of doing something like the following in other languages

```

val myList = List.instanceOf(3 , 4, 5);

```

In Scala, you can do

```

val myList = List.apply(3,4,5)

```

as the List object has an "apply" function as shown below. "**A***" means type "**A**", which can be an Int, String, Boolean, Any, etc and "*" repeated anynumber of times. E.g. Int, Int, Int as in (3, 5, 7).

```

def apply[A](xs: A*): List[A]

```

This enables us to create a list in Scala as shown below, and the "apply(…)" method is implicitly invoked.

```

val myList = List(3,4,5)

```

Note: Always have the Scala API doco handy. E.g. http://www.scala-lang.org/api. If you look at the Scala APIs, you will see "apply" methods. E.g. Future object, Promise object, List object, and the list goes on.

More Scala Interview Q&As

1. Q6 – Q12 Scala basics interview Q&As on functions

2. Q59 – Q64 Higher Order Functions in Scala Interview Q&As

Popular Member Posts

- ♦ 11 Spring boot interview questions & answers

850 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

768 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

399 views

18 Java scenarios based interview Questions and Answers

387 views

♦ 7 Java debugging interview questions & answers

308 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

305 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

297 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

293 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

246 views

001B: ♦ Java architecture & design concepts interview questions & answers

204 views

Bio **Latest Posts**

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ 01: ♥ Scala Actors tutorial step by step

02: Q6 – Q12 Scala FP interview questions & answers ›

Posted in Scala Interview Q&As

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer