Register | Login | Logout | Contact Us

Java-Success.com

Industrial strength Java/JEE Career Companion to open more doors

search here … Go

Home | Java FAQs | 600+ Java Q&As | Career | Tutorials | Member | Why?

Can u Debug? | Java 8 ready? | Top X | Productivity Tools | Judging Experience?

Home › member-paid › 02: Q6 – Q12 Scala FP interview questions & answers

02: Q6 – Q12 Scala FP interview questions & answers

Posted on July 27, 2016 by Arulkumaran Kumaraswamipillai

This extends Q1 – Q5 Scala Functional Programming basics interview questions & answers

Q6. What is a curried function in Scala?
A6. **Currying** is the technique of transforming a function with multiple arguments into a function with just one argument, and the other arguments are curried.

Currying is when you break down a function that takes multiple arguments into a series of functions that take part of the arguments. Here is an example:

600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

open all | close all

- Ice Breaker Interview
- Core Java Interview (
- JEE Interview Q&A (3
- Pressed for time? Jav
- SQL, XML, UML, JSC
- Hadoop & BigData Int
- Java Architecture Inte
- Scala Interview Q&As
  - Scala way of codin
    - 01: Coding Scala
  - 01: ♥ Q1 – Q6 Sca
  - 02: Q6 – Q12 Scala
  - 03: Q13 – Q18 Sca
  - 04: Q19 – Q26 Sca
  - 05: Q27 – Q32 Sca
  - 06: Q33 – Q40 Sca
  - 07: Q41 – Q48 Sca
  - 08: Q49 – Q58 Sca
  - 09: Q59 – Q65 Hig
  - 10: Q66 – Q70 Pat
  - 11: Q71 – Q77 – S

```
package com.mytutorial

object CurryingInScala extends App {

   //normal function
   def add(x: Int, y: Int): Int = {
     return x + y
   }

   println(add(5, 6)) // 11

   //curried function by separating out the argu
   def addCurried(x: Int)(y: Int): Int = {
     return x + y
   }

   println(addCurried(1) _) // returns a functio
   println((addCurried(1) _).apply(5)) // return
   println(addCurried(1)(2))   //returns 3

}
```

"(x: Int, y: Int)" is split into "(x: Int)(y: Int)".

Output:

```

11
<function1>
6
3

```

"_" is an anonymous function place holder parameter. The above curried function was written the shorter way, and can be written as shown below:

```

package com.mytutorial

object CurryingInScala extends App {

   //curried (i.e. partial) function by separati
   def addCurried(x: Int): (Int => Int) = {
      return (y: Int) => {   //return is optiona
         println("y = " + y);
         x + y
     }
   }

   println(addCurried(1)) // returns a function
   println((addCurried(1)).apply(5)) // returns
   println(addCurried(1)(2))   //returns 3
}

```

12: Q78 – Q80 Rec
⊞ Spring, Hibernate, & I
⊞ Testing & Profiling/Sa
⊞ Other Interview Q&A f
⊞ ▶ Free Java Intervie

16 Technical Key Areas

open all | close all
⊞ Best Practice (6)
⊞ Coding (26)
⊞ Concurrency (6)
⊞ Design Concepts (7)
⊞ Design Patterns (11)
⊞ Exception Handling (3
⊞ Java Debugging (21)
⊞ Judging Experience In
⊞ Low Latency (7)
⊞ Memory Managemen
⊞ Performance (13)
⊞ QoS (8)
⊞ Scalability (4)
⊞ SDLC (6)
⊞ Security (13)
⊞ Transaction Managen

80+ step by step Java Tutorials

open all | close all
⊞ Setting up Tutorial (6)
⊞ Tutorial - Diagnosis (2
⊞ Akka Tutorial (9)
⊞ Core Java Tutorials (2
⊞ Hadoop & Spark Tuto
⊞ JEE Tutorials (19)
⊞ Scala Tutorials (1)

Output:

```

<function1>
6
3

```

The curried function can be simplified as shown below

```

package com.mytutorial

object CurryingInScala extends App {

   //curried (i.e. partial) function by separati
   def addCurried(x: Int) = (y: Int) => x + y

   println(addCurried(1)) // returns a function
   println((addCurried(1)).apply(5)) // returns
   println(addCurried(1)(2))   //returns 3
}

```

Q7. What is a closure in Scala?

A7. A **closure** is a function, whose return value depends on the value of one or more variables declared outside this function.

```

package com.mytutorial

object ClosuresInScala extends App{

   val more = 20;

   //curried function has a reference to variabl
   //outside the function but in the enclosing s
   def addCurried(x: Int) = (y: Int) => x + y +

   println(addCurried(1)(2)) //returns 23
}

```

Q8. Can you upply the function place holder "_" (i.e. underscore) to the following code snippet?

```

  val listOfNumbers: List[Int] = (1 to 10).toList
  val oddNumbers = listOfNumbers.filter(x => x %
```

⊞ Spring & HIbernate Tu
⊞ Tools Tutorials (19)
⊞ Other Tutorials (45)

100+ Java pre-interview coding tests

open all | close all
⊞ Can you write code?
⊞ ♦ Complete the given
⊞ Converting from A to I
⊞ Designing your classe
⊞ Java Data Structures
⊞ Passing the unit tests
⊞ What is wrong with th
⊞ Writing Code Home A
⊞ Written Test Core Jav
⊞ Written Test JEE (1)

How good are your .....?

open all | close all
⊞ Career Making Know-
⊞ Job Hunting & Resum

A8. Scala uses the underscore to mean different things in different contexts, but you can think of it as an unnamed wildcard. A place holder makes a functional syntax more concise as shown below.

```

  val listOfNumbers: List[Int] = (1 to 10).toList
  val oddNumbers = listOfNumbers.filter(_ % 2 !=

```

Q9. What will be the out of the following **partial function**?

```

package com.mytutorial

object PartialFunctionInScala  extends App {

  def add(x: Int, y: Int, z: Int) = x +y + z

  val addAnother = add(1, _:Int, _: Int)

  println(addAnother)
  println(addAnother(6, 2))

}

```

A9. You can partially apply a function with an **underscore**, which gives you another function. The "_" here stands for any argument.

Output:

```

<function2>
9

```

Q10. Can you write a capitalize function in Scala that takes variable number of Strings input as shown below?

capitalize("john", "sam", "peter")
capitalize("apple", "pears")

A10. The special syntax with “*” is shown below to take variable length arguments in Scala.

```

package com.mytutorial

import scala.collection.mutable.ArrayBuffer

object VariableArgumentsFunctionInScala extends

  //return type is Seq[String]
  def capitalize (input: String*): Seq[String] =
    input.map (element => element.capitalize)
  }

  println(capitalize("john", "sam", "peter"))
  println(capitalize("apple", "pears"))
}

```

Output:

```

ArrayBuffer(John, Sam, Peter)
ArrayBuffer(Apple, Pears)

```

Q11. Can you write a recursive function in Scala to calculate the factorial?

A11.

```

package com.mytutorial

object RecursiveFunctionInScala {

  //Unit means void function
  def main (args: Array[String]) : Unit = {
    println(factorial(4))
  }

  def factorial (input: Int): BigInt = {

    if(input == 0) {
      return BigInt(1);
    } else {
      return input * factorial(input -1) // recu
    }
  }
}

```

First Pass: 4 * factorial(3)
Second Pass: 3 * factorial(2)
Third Pass: 2 * factorial(1)
Fourth Pass: 1 * 1 [Exit Condition is Reached]

The result is: 4 * 3 * 2 * 1 * 1 = 24

Q12. What is tail recursion? Can you write a tail recursion function Scala to calculate the factorial?

A12. A recursive call to be **tail** recursive, the call back to the function must be the last action performed in that function. In the above example, since the result of each recursive call is being multiplied by the next recursion call, the recursive call is NOT the last action performed in the function, hence NOT a tail recursion.

To make it a tail recursion, the “factorial” function needs to take two arguments. The “interimResult” argument stores the intermediate result, and it is initialized from 1.

```

package com.mytutorial

object RecursiveFunctionInScala {

  def main(args: Array[String]): Unit = {
    println(factorial(1, 4)) // 1 is initial int
  }

  //tail recursion
  def factorial(intermediateResult: BigInt, inpu

    if (input == 0) {
      return intermediateResult;
    }

    //recursion is the last call. NO "input * fa
    factorial(intermediateResult * input, input

  }
}

```

Q. What is the benefit of tail recursion?

A. Since recursion takes place in the stack memory, and in the recursion example the result of each call must be remembered (e.g. 4 * 3 * factorial(2) …), each recursive call requires an entry on the stack until all recursive calls have been made. This makes the recursive call more memory intensive. The “tail recursion” stores the result as an intermediate result.

Q. Since in Scala you can declare a function within another function, is it possible for the factorial method to take a single argument as in factorial(4)?
A. Yes.

```

package com.mytutorial

object RecursiveFunctionInScala {

  def main(args: Array[String]): Unit = {
    println(factorial(56))
  }

  def factorial(input: Int): BigInt = {
    //tail recursion
    def factorialWithIntermediateResult(intermed
      if (input == 0) {
        return intermediateResult;
      }

      //recursion is the last call. NO "input *
      factorialWithIntermediateResult(intermedia
    }

    factorialWithIntermediateResult(1, input)
  }
}

```

Popular Member Posts

♦ 11 Spring boot interview questions & answers

850 views

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

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

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

Posted in member-paid, 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

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.