

Scala Interview Questions and Answers

In this section, we will pickup each and every question from above list and discuss in-detail with suitable examples(if required). If you want to understand these concepts in-depth with examples, please go through my previous posts in Scala Tutorials section.

1. What is Scala? Is it a Language or Platform? Does it support OOP or FP? Who is the father of Scala?

Scala stands for SCALable LAnguage. Martin Odersky is the father of Scala.

Scala is a Multi-Paradigm Programming Language, which supports both Object-Oriented and Functional Programming concepts. It is designed and developed by Martin Odersky.

Scala is a Type-Safe Object-Functional Programming JVM Language. Scala runs on JVM(Java Virtual Machine).

Scala is a Hybrid Functional (Object-Oriented and Functional) Programming JVM Language. Scala has a Strong and Statically Type System. In Scala, all types are checked at compile-time.

2. Is Scala Statically-Typed Language? What is Statically-Typed Language and What is Dynamically-Typed Language? What is the difference between statically typed and dynamically typed languages?

Yes, Scala is a Statically-Typed Language.

Statically-Typed Language means that Type checking is done at compile-time by compiler, not at run-time. The main Advantage of these kinds of Languages is: As a Developer, we should care about writing right code to avoid all compile-time errors. As Compiler checks many of the errors at compile-time, we don't get much issues or bugs at run-time.

Examples:- Java, Scala, C, C++, Haskell etc.

Dynamically-Typed Language means that Type checking is done at run-time, not at compile-time by compiler. As a compiler won't check any type checking at compile-time, We can expect more run-time issues or bugs.

Example:- Groovy, JavaScript, Ruby, Python, Smalltalk etc.

3. Is Scala a Pure OOP Language? Is Java a Pure OOP Language?

Pure Object-Oriented Programming Language means that everything should be an Object.

Java is not a Pure Object-Oriented Programming (OOP) Language because it supports the following two Non-OOP concepts:

Java supports primitive data types. They are not objects.

Java supports Static members. They are not related to objects.

Yes, Scala is a Pure Object-Oriented Programming Language because in Scala, everything is an Object and everything is a value. Functions are values and values are Objects.

Scala does not have primitive data types and also does not have static members.

4. Does Scala support all Functional Programming concepts? Does Java 8 support all Functional Programming concepts?

Yes, Scala supports all Functional Programming (FP) concepts. Java 8 has introduced some Functional Programming constructs, but it does NOT support all Functional Programming concepts.

For instance, Java 8 does not support Pattern Matching, Function Currying, Implicits etc.

5. What are the major advantages of Scala Language? Are there any drawbacks of Scala Language?

If we use Scala Language to develop our applications, we can get the following benefits or advantages and drawbacks:

Advantages of Scala Language:-

Simple and Concise Code

Very Expressive Code

More Readable Code

100% Type-Safe Language

Immutability and No Side-Effects

More Reusable Code

More Modularity

Do More With Less Code

Very Flexible Syntax

Supports all OOP Features

Supports all FP Features. Highly Functional.

Less Error Prone Code

Better Parallel and Concurrency Programming
Highly Scalable and Maintainable code
Highly Productivity
Distributed Applications
Full Java Interoperability
Powerful Scala DSLs available
REPL to learn Scala Basics
Drawbacks of Scala Language:-

Less Readable Code
Bit tough to Understand the Code for beginners
Complex Syntax to learn
Less Backward Compatibility
NOTE:- We can write Scala Code either more readable or less readable way.

6. What is the Main drawback of Scala Language?

Apart from many benefits of Scala, it has one major Drawback: Backward Compatibility Issue. If we want to upgrade to latest version of Scala, then we need to take care of changing some package names, class names, method or function names etc.

For instance, If you are using old Scala version and your project is using BeanProperty annotation. It was available in “scala.reflect” like “scala.reflect.BeanProperty” in old versions. If we want to upgrade to new Scala versions, then we need to change this package from “scala.reflect” to “scala.beans”.

7. What is the main motto of Scala Language?

Like Java’s Motto “Write Once Run Anywhere”, Scala has “Do More With Less” or “Do More With Less Code” Motto. “Do More With Less” means that we can develop more complex program or logic with less code.

8. What are the popular JVM Languages available now?

Java, Scala, Groovy and Closure are most popular JVM (Java Virtual Machine) languages.

Scala, Groovy and Closure JVM languages supports both Object-Oriented Programming Features and Functional Programming Features.

Java SE 8 supports all Object-Oriented Programming Features. However, it supports very few Functional Programming Features like Lambda Expressions, Functions, Type Inference, Higher-Order Functions.

9. Like Java’s java.lang.Object class, what is the super class of all classes in Scala?

As we know in Java, the super class of all classes (Java API Classes or User Defined Classes) is java.lang.Object. In the same way in Scala, the super class of all classes or traits is “Any” class.

Any class is defined in scala package like “scala.Any”.

10. What is default access modifier in Scala? Does Scala have “public” keyword?

In Scala, if we don’t mention any access modifier to a method, function, trait, object or class, the default access modifier is “public”. Even for Fields also, “public” is the default access modifier.

Because of this default feature, Scala does not have “public” keyword.

11. What is “Type Inference” in Scala?

Types can be inferred by the Scala Compiler at compile-time. It is known as “Type Inference”. Types means Data type or Result type. We use Types at many places in Scala programs like Variable types, Object types, Method/Function Parameter types, Method/Function return types etc.

In simple words, determining the type of a variable or expression or object etc at compile-time by compiler is known as “Type Inference”.

12. What are the similarities and differences between Scala’s Int and Java’s java.lang.Integer? What is the relationship between Int and RichInt in Scala?

Similarities between Scala’s Int and Java’s java.lang.Integer:

Both are classes.
Both are used to represent integer numbers.
Both are 32-bit signed integers.

Differences between Scala's Int and Java's java.lang.Integer:

Scala's Int class does not implement Comparable interface.

Java's java.lang.Integer class implements Comparable interface.

Java's Integer is something similar to Scala's Int and RichInt. RichInt is a final class defined in scala.runtime package like "scala.runtime.RichInt".

In Scala, the Relationship between Int and RichInt is that when we use Int in a Scala program, it will automatically convert into RichInt to utilize all methods available in that Class. We can say that RichInt is an Implicit class of Int. (We will discuss "What is Implicit and the advantages of Implicits in my next post).

13. What is Nothing in Scala? What is Nil in Scala? What is the relationship between Nothing and Nil in Scala?

In Scala, Nothing is a Type (final class). It is defined at the bottom of the Scala Type System that means it is a subtype of anything in Scala. There are no instances of Nothing.

Use Cases of Nothing In Scala:-

If Nothing does not have any instances, then when do we use this one in Scala Applications?

Nil is defined using Nothing (See below for example).

None is defined using Nothing.

object None extends Option[Nothing]

We can use Nothing as a return type of methods which never return.

We can use Nothing as a return type of methods which terminates abnormally.

Nil is an object, which is used to represent an empty list. It is defined in "scala.collection.immutable" package as shown below:

```
object Nil extends List[Nothing]
```

Example:-

```
scala> Nil
```

```
res5: scala.collection.immutable.Nil.type = List()
```

```
scala> Nil.length
```

```
res6: Int = 0
```

14. What is Null in Scala? What is null in Scala? What is difference between Null and null in Scala?

Null is a Type (final class) in Scala. Null type is available in "scala" package as "scala.Null". It has one and only one instance that is null.

In Scala, "null" is an instance of type scala.Null type.

Example:-

```
scala> val myNullRef : Null = null
```

```
myNullRef: Null = null
```

We cannot assign other values to Null type references. It accepts only 'null' value.

Null is a subtype of all Reference types. Null is at the bottom of the Scala Type System. As it is NOT a subtype of Value types, we can assign "null" to any variable of Value type.

Example:-

```
scala> val myInt : Int = null
```

```
:10: error: an expression of type Null is ineligible for implicit conversion
```

```
val myInt : Int = null
```

^

Here type mismatch error. found : Null(null) but required: Int. The implicit conversions between Null and Int are not applicable because they are ambiguous.

15. What is Unit in Scala? What is the difference between Java's void and Scala's Unit?

In Scala, Unit is used to represent "No value" or "No Useful value". Unit is a final class defined in "scala" package that is "scala.Unit".

Unit is something similar to Java's void. But they have few differences.

Java's void does not any value. It is nothing.

Scala's Unit has one value ()

() is the one and only value of type Unit in Scala. However, there are no values of type void in Java.

Java's void is a keyword. Scala's Unit is a final class.

Both are used to represent a method or function is not returning anything.

16. What is the difference between val and var in Scala?

In Scala, both `val` and `var` are used to define variables. However, they have some significant differences.

`var` stands for variable.

`val` stands for value.

As we know, variable means changeable and value means constant.

`var` is used to define Mutable variables that means we can reassign values once its created.

`val` is used to define Immutable variables that means we cannot reassign values once its created.

In simple Java terminology, `var` means 'variable' and `val` means 'final variable'.

17. What is REPL in Scala? What is the use of Scala's REPL? How to access Scala REPL from CMD Prompt?

REPL stands for Read-Evaluate-Print Loop. We can pronounce it as 'ripple'. In Scala, REPL is acts as an Interpreter to execute Scala code from command prompt. That's why REPL is also known as Scala CLI(Command Line Interface) or Scala command-line shell.

The main purpose of REPL is that to develop and test small snippets of Scala code for practice purpose. It is very useful for Scala Beginners to practice basic programs.

We can access REPL by using "scala" command. When we type "scala" command at CMD Prompt, we will get REPL shell where we can type and execute scala code.

```
D:\> scala
```

```
scala>
```

18. What are the Scala Features?

Scala Language supports the following features:

Supports both OOP-style(Imperative-Style) and Functional-Style Programming

Pure Object-Oriented Programming Language

Supports all Functional Features

REPL(Read-Evaluate-Print Loop) Interpreter

Strong Type System

Statically-Typed Language

Type Inference

Supports Pattern Matching

Supports Closures

Supports Persistent Data Structures

Uses Actor Model to develop Concurrency Applications

Interoperable with Java

Available all development tools – IDEs, Build Tools, Web Frameworks, TDD and BDD Frameworks

19. How do we implement loops functionally? What is the difference between OOP and FP style loops?

We know how to implement loops in Object-Oriented style: Using Mutable Temporary variables, update the variable value and use Loop constructs. It is very tedious and unsafe approach. It is not Thread-Safe.

Object-Oriented style uses the following constructs to implement Loops:

Loop Constructs

Mutability

Side Effects

We can implement same Loops differently in Functional way. It is Thread-Safe. We can use the following two techniques to implement the loops in functional style:

Recursion

Tail-Recursion

Immutability

No Side-Effects

20. What is "Application" in Scala or What is Scala Application? What is "App" in Scala? What is the use of Scala's App?

Scala Application:

In Scala, App is a trait defined in scala package like "scala.App". It defines main method. If an Object or a Class extends this trait, then they will become as Scala Executable programs automatically because they will inherit main method from Application.

The main advantage of using App is that we don't need to write main method. The main drawback of using App is that we should use same name "args" to refer command line argument because scala.App's main() method uses this name.

Example:-

Without Scala App:

```
object MyApp {  
  def main( args: Array[String]){  
    println("Hello World!")  
  }  
}
```

With Scala App:

```
object MyApp extends App{  
  println("Hello World!")  
}
```

If we observe above two examples, in second example we have not defined main method because we have inherited from Scala App(Application).

Before Scala 2.9, we have scala.Application trait. But it is deprecated by scala.App since Scala 2.9 version.

21. Does Scala support Operator Overloading? Does Java support Operator Overloading?

Java does not support Operator Overloading. Scala supports Operator Overloading.

The reason is that Java does not want to support some misleading method names like “+*/”. Scala has given this flexibility to Developer to decide which methods/functions name should use.

When we call 2 + 3 that means ‘+’ is not an operator, it is a method available in Int class (or it’s implicit type). Internally, this call is converted into “2.+(3)”.

22. What is an Expression? What is a Statement? Difference between Expression and Statement?

Expression:

Expression is a value that means it will evaluate to a Value. As an Expression returns a value, We can assign it to a variable.

Example:- Scala’s If condition, Java’s Ternary operator.

Statement:

Statement defines one or more actions or operations. That means Statement performs actions. As it does not return a value, we cannot assign it to a Variable.

Example:- Java’s If condition.

23. What is the difference between Java’s “If..Else” and Scala’s “If..Else”?

Java’s “If..Else”:

In Java, “If..Else” is a statement, not an expression. It does not return a value and cannot assign it to a variable.

Example:-

```
int year;  
if( count == 0)  
  year = 2014;  
else  
  year = 2015;
```

Scala’s “If..Else”:

In Scala, “If..Else” is an expression. It evaluates a value i.e. returns a value. We can assign it to a variable.

```
val year = if( count == 0) 2014 else 2015
```

NOTE:-Scala’s “If..Else” works like Java’s Ternary Operator. We can use Scala’s “If..Else” like Java’s “If..Else” statement as shown below:

```
val year = 0  
if( count == 0)  
  year = 2014  
else  
  year = 2015
```

24. Is Scala an Expression-Based Language or Statement-Based Language? Is Java an Expression-Based Language or Statement-Based Language?

In Scala, everything is a value. All Expressions or Statements evaluates to a Value. We can assign Expression, Function, Closure, Object etc. to a Variable. So Scala is an Expression-Oriented Language.

In Java, Statements are not Expressions or Values. We cannot assign them to a Variable. So Java is not an Expression-Oriented Language. It is a Statement-Based Language.

25. Tell me some features which are supported by Java, but not by Scala and Vice versa?

Java does not support Operator Overloading, but Scala supports it.

Java supports ++ and — operators , but Scala does not support them.

Java has Checked and Unchecked Exceptions, but Scala does not have Checked Exceptions.

Scala does not support break and continue statements, but Java uses them.

Scala does not have explicit Type casting, but Java supports this feature.

Scala supports Pattern Matching, but Java does not.

Java uses Primitive Data types, but Scala does not have.

Java supports static members, but Scala does not have static members concept.

Scala supports Implicits and Traits, Java does not support them.

NOTE:-This list goes beyond one page. However, these are some important points to remember about differences in Scala and Java features to face Scala Interviews.

26. What is the difference between Function and Method in Scala?

Scala supports both functions and methods. We use same syntax to define functions and methods, there is no syntax difference.

However, they have one minor difference:

We can define a method in a Scala class or trait. Method is associated with an object (An instance of a Class). We can call a method by using an instance of a Class. We cannot use a Scala Method directly without using object.

Function is not associated with a class or trait. It is defined in a Scala Package. We can access functions without using objects, like Java's Static Methods.

NOTE:- We will discuss about Class, Trait,Package, Object etc in my coming posts.

27. How many public class files are possible to define in Scala source file?

In Java, we can define at-most one public class/interface in a Source file. Unlike Java, Scala supports multiple public classes in the same source file.

We can define any number of public classes/interfaces/traits in a Scala Source file.

28. Like Java, what are the default imports in Scala Language?

We know, java.lang is the default package imported into all Java Programs by JVM automatically. We don't need to import this package explicitly.

In the same way, the following are the default imports available in all Scala Programs:

```
java.lang package
scala package
scala.PreDef
```

29. How many operators are there in Scala and Why?

Unlike Java and like C++, Scala supports Operator Overloading. Scala has one and only operator that is “=” (equalto) operator. Other than this all are methods only.

For instance 2 + 3, here “+” is not an Operator in Scala. “+” is method available in Int class. Scala Compiler observes 2 and 3 are Integers and tries to find that “+” method in Int class. So Scala Compiler converts “2 + 3” expression into “2.+(3)” and make a call to “+” method on integer object “2” and pass integer object “3” as parameter to “+” method.

Both “2 + 3” and “2.+(3)” are equal. It's just Scala's syntactic sugar to write programs in Functional style.

30. Mention Some keywords which are used by Java and not required in Scala? Why Scala does not require them?

Java uses the following keywords extensively:

‘public’ keyword – to define classes, interfaces, variables etc.

‘static’ keyword – to define static members.

Scala does not required these two keywords. Scala does not have ‘public’ and ‘static’ keywords.

In Scala, default access modifier is ‘public’ for classes,traits, methods/functions, fields etc. That’s why, ‘public’ keyword is not required.

To support OOP principles, Scala team has avoided ‘static’ keyword. That’s why Scala is a Pure-OOP Language. It is very tough to deal static members in Concurrency applications.

31. What is PreDef in Scala? What is the main purpose of PreDef in Scala?

In Scala, PreDef is an object defined in scala package as “scala.PreDef”. It is an utility object.

It defines many utility methods as shown below:

Console IO (print,println etc)

Collection utility methods

String utility methods

Implicit conversion methods

Assertion utility methods etc.

For instance, print, println, readLine, readInt, require etc methods are defined in PreDef object.

In Scala, PreDef is available to use its methods without importing in all Scala Programs because Scala Compiler imports this object into all compilation units like Class, Object, Trait etc. automatically.

=====

32 . What is Primary Constructor? What is Secondary or Auxiliary Constructor in Scala? What is the purpose of Auxiliary Constructor in Scala? Is it possible to overload constructors in Scala?

Scala has two kinds of constructors:

Primary Constructor

Auxiliary Constructor

Primary Constructor

In Scala, Primary Constructor is a constructor which is defined with class definition itself. Each class must have one Primary Constructor: Either Parameter constructor or Parameterless constructor.

Example:-

```
class Person{ }
```

Above Person class has one Zero-parameter or No-Parameter or Parameterless Primary constructor to create instances of this class.

```
class Person (firstName: String, lastName: String){ }
```

Above Person class has a two Parameters *Primary constructor* to create instances of this class.

Auxiliary Constructor is also known as Secondary Constructor. We can declare a Secondary Constructor using ‘def’ and ‘this’ keywords as shown below:

```
class Person (firstName: String, middleName:String, lastName: String){  
  def this(firstName: String, lastName: String){  
    this(firstName, "", lastName)  
  }  
}
```

33. What is the use of Auxiliary Constructors in Scala? Please explain the rules to follow in defining Auxiliary Constructors in Scala?

In Scala, The main purpose of Auxiliary Constructors is to overload constructors. Like Java, We can provide various kinds of constructors so that use can choose the right one based on his requirement.

Auxiliary Constructor Rules:

They are like methods only. Like methods, we should use 'def' keyword to define them.
We should use same name 'this' for all Auxiliary Constructors.
Each Auxiliary Constructor should start with a call to previous defined another Auxiliary Constructor or Primary Constructor.
Otherwise compile-time error.
Each Auxiliary Constructor should differ with their parameters list: may be by number or types.
Auxiliary Constructors cannot call a super class constructors. They should call them through Primary Constructor only.
All Auxiliary Constructors call their Primary Constructor either directly or indirectly through other Auxiliary Constructors.
NOTE:- If you want to learn about Scala's Constructors, please refer my Scala posts at: Primary Constructor and Auxiliary Constructor.

34. What are the differences between Array and ArrayBuffer in Scala?

Differences between Array and ArrayBuffer in Scala:

Array is fixed size array. We cannot change its size once its created.
ArrayBuffer is variable size array. It can increase or decrease it's size dynamically.
Array is something similar to Java's primitive arrays.
ArrayBuffer is something similar to Java's ArrayList.

35. What is case class? What is case object? What are the Advantages of case class?

Case class is a class which is defined with "case class" keywords. Case object is an object which is defined with "case object" keywords. Because of this "case" keyword, we will get some benefits to avoid boilerplate code.

We can create case class objects without using "new" keyword. By default, Scala compiler prefixes "val" for all constructor parameters. That's why without using val or var, Case class's constructor parameters will become class members, it is not possible for normal classes.

Advantages of case class:

By default, Scala Compiler adds toString, hashCode and equals methods. We can avoid writing this boilerplate code.
By default, Scala Compiler adds companion object with apply and unapply methods that's why we don't need new keyword to create instances of a case class.
By default, Scala Compiler adds copy method too.
We can use case classes in Pattern Matching.
By default, Case class and Case Objects are Serializable.

36. What is the difference between Case Object and Object(Normal Object)?

Normal object is created using "object" keyword. By default, It's a singleton object.

object MyNormalObject

Case Object is created using "case object" keywords. By default, It's also a singleton object

case object MyCaseObject

By Default, Case Object gets toString and hashCode methods. But normal object cannot.
By Default, Case Object is Serializable. But normal object is not.

37. When compare to Normal Class, What are the major advantages or benefits of a Case-class?

The following are the major advantages or benefits of a Case class over Normal Classes:

Avoids lots of boiler-plate code by adding some useful methods automatically.
By default, supports Immutability because it's parameters are 'val'
Easy to use in Pattern Matching.
No need to use 'new' keyword to create instance of Case Class.
By default, supports Serialization and Deserialization.

38. What is the usage of isInstanceOf and asInstanceOf methods in Scala? Is there anything similar concept available in Java?

Both isInstanceOf and asInstanceOf methods are defined in Any class. So no need import to get these methods into any class or object.

"isInstanceOf" method is used to test whether the object is of a given type or not. If so, it returns true. Otherwise returns false.

```
scala> val str = "Hello"
```

```
scala> str.isInstanceOf[String]  
res0: Boolean = false
```


“asInstanceOf” method is used to cast the object to the given a type. If the given object and type are of same type, then it cast to given type. Otherwise, it throws java.lang.ClassCastException.

```
scala> val str = "Hello".asInstanceOf[String]
str: String = Hello
```

In Java, ‘instanceof’ keyword is similar to Scala’s ‘asInstanceOf’ method. In Java, the following kind of manual type casting is similar to Scala’s ‘asInstanceOf’ method.

```
AccountService service = (AccountService)
context.getBean("accountService");
```

39. How do you prove that by default, Case Object is Serializable and Normal Object is not?

Yes, By Default, Case Object is Serializable. But normal object is not. We can prove this by using isInstanceOf method as shown below:

```
scala> object MyNormalObject
defined object MyNormalObject
```

```
scala> MyNormalObject.isInstanceOf[Serializable]
res0: Boolean = false
```

```
scala> case object MyCaseObject
defined object MyCaseObject
```

```
scala> MyCaseObject.isInstanceOf[Serializable]
res1: Boolean = true
```

40. Difference between Array and List in Scala?

Arrays are always Mutable where as List is always Immutable.

Once created, We can change Array values where as we cannot change List Object.

Arrays are fixed-size data structures where as List is variable-sized data structures. List’s size is automatically increased or decreased based on it’s operations we perform on it.

Arrays are Invariants where as Lists are Covariants.

NOTE:- If you are not sure about Invariant and Covariant, please read my next post on Scala Interview Questions.

41. What is the difference between “val” and “lazy val” in Scala? What is Eager Evaluation? What is Lazy Evaluation?

As we discussed in my Basic Scala Interview Questions, “val” means value or constant which is used to define Immutable variables.

There are two kinds of program evaluations:

Eager Evaluation

Lazy Evaluation

Eager Evaluation means evaluating program at compile-time or program deployment-time irrespective of clients are using that program or not.

Lazy Evaluation means evaluating program at run-time on-demand that means when clients access the program then only its evaluated.

The difference between “val” and “lazy val” is that “val” is used to define variables which are evaluated eagerly and “lazy val” is also used to define variables but they are evaluated lazily.

42. What is the Relationship between equals method and == in Scala? Differentiate Scala’s == and Java’s == Operator?

In Scala, we do NOT need to call equals() method to compare two instances or objects. When we compare two instances with ==, Scala calls that object’s equals() method automatically.

Java’s == operator is used to check References Equality that is whether two references are pointing to the same object or not. Scala’s == is used to check Instances Equality that is whether two instances are equal or not.

43. Difference between Scala’s Inner class and Java’s Inner class?

In Java, Inner class is associated with Outer class that is Inner class a member of the Outer class.

Unlike Java, Scala treats the relationship between Outer class and Inner class differently. Scala’s Inner class is associated with Outer class object.

44. What is Diamond Problem? How Scala solves Diamond Problem?

A Diamond Problem is a Multiple Inheritance problem. Some people call this problem as Deadly Diamond Problem.

In Scala, it occurs when a Class extends more than one Traits which have same method definition as shown below.

diamond_problem

Unlike Java 8, Scala solves this diamond problem automatically by following some rules defined in Language. Those rules are called “Class Linearization”.

Example:-

```
trait A{
  def display(){ println("From A.display") }
}
trait B extends A{
  override def display() { println("From B.display") }
}
trait C extends A{
  override def display() { println("From C.display") }
}
class D extends B with C{ }
```

```
object ScalaDiamondProblemTest extends App {
  val d = new D
  d.display
}
```

Here output is “From C.display” from trait C. Scala Compiler reads “extends B with C” from right to left and takes “display” method definition from left most trait that is C.

NOTE:- See my post on “Scala Traits in Depth” to know this with clear explanation.

45. Why Scala does NOT have “static” keyword? What is the main reason for this decision?

As we know, Scala does NOT have “static” keyword at all. This is the design decision done by Scala Team.

The main reason to take this decision is to make Scala as a Pure Object-Oriented Language. “static” keyword means that we can access that class members without creating an object or without using an object. This is completely against with OOP principles.

If a Language supports “static” keyword, then that Language is not a Pure Object-Oriented Language. For instance, as Java supports “static” keyword, it is NOT a Pure Object-Oriented Language. But Scala is a Pure Object-Oriented Language.

46. What is the use of “object” keyword in Scala? How to create Singleton objects in Scala?

In Scala, object keyword is used for the following purposes:

It is used to create singleton object in Scala.

```
object MySingletonObject
```

Here, MySingletonObject becomes singleton object automatically.

object keyword is used to define Scala Applications that are executable Scala programs.

```
object MyScalaExecutableProgram{
  def main(args: Array[String]){
    println("Hello World")
  }
}
```

When we define main method in object as shown above (its same as main() method in Java), it becomes automatically as an executable Scala program.

It is used to define static members like static variables and static methods without using ‘static’ keyword.

```
object MyScalaStaticMembers{
  val PI: Double = 3.1414
  def add(a: Int, b: Int) = a + b
}
```

By def PI variable and add methods will become as static members. That means we can call them without creating a separate object like `MyScalaStaticMembers.add(10,20)`.

It is used to define Factory methods. Please see my next question about this.

How to define Factory methods using object keyword in Scala? What is the use of defining Factory methods in object?

In Scala, we use 'object' keyword to define Factory methods. The main purpose of these Factory methods in Scala is to avoid using 'new' keyword. Without using 'new' keyword we can create objects.

To define Factory methods:

We can use apply method to define Factory methods in Scala. If we have Primary Constructor and Multiple Auxiliary constructors, then we need to define multiple apply methods as shown below.

```
class Person(val firstName: String, val middleName: String, val lastName: String){
  def this(firstName: String, lastName: String){
    this(firstName,"",lastName)
  }
}
object Person{
  def apply(val firstName: String, val middleName: String, val lastName: String)
    = new Person(firstName,middleName,lastName)

  def apply(val firstName: String, val lastName: String)
    = new Person(firstName, lastName)
}
```

Now we can create Person objects without using new keyword or with new keyword upto your wish.

```
val p1 = new Person("Scala","Java")
or
val p1 = Person("Scala","Java")
```

47. What is apply method in Scala? What is unapply method in Scala? What is the difference between apply and unapply methods in Scala?

In Scala, apply and unapply methods play very important role. They are also very useful in Play Framework in mapping and unmapping data between Form data and Model data.

In simple words,

apply method: To compose or assemble an object from it's components.

unapply method: To decompose or dis-assemble an object into it's components.

Scala's apply method:

It is used to compose an object by using its components. Suppose if we want to create a Person object, then use firstName and laststName two components and compose Person object as shown below.

```
class Person(val firstName: String, val lastName: String)
```

```
object Person{
  def apply(firstName: String, lastName: String)
    = new Person(firstName, lastName)
}
```

Scala's unapply method:

It is used to decompose an object into its components. It follows reverse process of apply method. Suppose if we have a Person object, then we can decompose this object into it's two components: firstName and laststName as shown below.

```
class Person(val firstName: String, val lastName: String)
```

```
object Person{
  def apply(firstName: String, lastName: String)
    = new Person(firstName, lastName)

  def unapply(p: Person): (String,String)
    = (p.firstName, p.lastName)
}
```

48. How does it work under-the-hood, when we create an instance of a Class without using 'new' keyword in Scala? When do we go for this approach? How to declare private constructors in Scala?

In Scala, when we create an instance of a Class without using 'new' keyword, internally it make a call to appropriate apply method available in Companion object. Here appropriate apply method means that matched with parameters.

When do we choose this option: When we need to provide private constructor and we need to avoid using 'new' keyword, we can implement only apply method with same set of parameters and allow our class users to create it without new keyword.

49. How do we declare a private Primary Constructor in Scala? How do we make a call to a private Primary Constructor in Scala?

In Scala, we can declare a private Primary Constructor very easily. Just define a Primary Constructor as it is and add 'private' just after class name and before parameter list as shown below:

```
class Person private (name: String)
object Person{
  def apply(name: String) = new Person(name)
}
```

As it's a private constructor, we cannot call it from outside. We should provide a factory method (that is apply method) as shown above and use that constructor indirectly.

50. Does a Companion object access private members of it's Companion class in Scala?

Generally, private members means accessible only within that class. However Scala's Companion class and Companion Object has provided another feature.

In Scala, a Companion object can access private members of it's Companion class and Companion class can access it's Companion object's private members.

51. What is the main design decision about two separate keywords: class and object in Scala? How do we define Instance members and Static members in Scala?

In Scala, we use class keyword to define instance members and object keyword to define static members. Scala does not have static keyword, but still we can define them by using object keyword.

The main design decision about this is that the clear separation between instance and static members. Loosely coupling between them. And other major reason is to avoid static keyword so that Scala will become a Pure-OOP Language.

52. What is object in Scala? Is it a singleton object or instance of a class?

Unlike Java, Scala has two meanings about 'object'. Don't get confuse about this, I will explain it clearly. In Java, we have only one meaning for object that is "An instance of a class".

Like Java, the first meaning of object is "An instance of a class".

```
val p1 = new Person("Scala","Java")
or
val p1 = Person("Scala","Java")
```

Second meaning is that object is a keyword in Scala. It is used to define Scala Executable programs, Companion Objects, Singleton Objects etc.

53. What is a Companion Object in Scala? What is a Companion Class in Scala? What is the use of Companion Object in Scala?

In simple words, if a Scala class and object shares the same name and defined in the same source file, then that class is known as "Companion Class" and that object is known as "Companion Object".

When we create a Class by using Scala "class" keyword and Object by using Scala "object" keyword with same name and within the same source file, then that class is known as "Companion Class" and that object is known as "Companion Object".

Example:-
Employee.scala

```
class Employee{ }
object Employee{ }
```

In Scala, The main purpose of Companion Object is to define apply methods and avoid using new keyword in creating an instance of that Companion class object.

54. How to implement interfaces in Scala?

As we know from Java background, we use interface to define contract.

However, there is no interface concept in Scala. Even, Scala doesn't have interface keyword. Scala has a more powerful and flexible concept i.e. trait for this purpose.

55. What is Range in Scala? How to create a Range in Scala?

Range is a Lazy Collection in Scala. Range is a class available in 'scala' package like 'scala.Range'. It is used to represent a sequence of integer values. It is an ordered sequence of integers.

Example:-

```
scala> 1 to 10
res0: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> 1 until 10
res1: scala.collection.immutable.Range = Range(1, 2, 3, 4, 5, 6, 7, 8, 9)
How many values of type Nothing have in Scala?
```

In Scala, Nothing type have no values that is zero. It does not have any values. It is a subtype of all Value classes and Reference classes.

56. How many values of type Unit have in Scala?

In Scala, Unit is something similar to Java's void keyword. It is used to represent "No value exists". It has one and only one value that is ().

57. What is a pure function?

A pure function is a function without any observable side-effects. That means it returns always same results irrespective how many times we call it with same inputs.

A pure function always gives same output for the same inputs.

For Example:-

```
scala> 10 + 20
res0: Int = 30
scala>
scala> 10 + 20
res0: Int = 30
Here "+" a pure function available in Int class. It gives same result 30 for same inputs 10 and 20, irrespective how many times we call it.
```

58. In FP, What is the difference between a function and a procedure?

Both are used to perform computation, however they have one major difference in Functional Programming world.

A function is a computation unit without side-effect where as a Procedure is also a computation unit with side-effects.

59. What are the major differences between Scala's Auxiliary constructors and Java's constructors?

Scala's Auxiliary constructor is almost similar to Java's constructor with few differences.

Compared to Java's constructors, Auxiliary constructors have the following few differences:

The auxiliary constructors are called using "this" keyword.

All auxiliary constructor are defined with the same name that is "this". In Java, we use class name to define constructors.

Each auxiliary constructor must start with a call to a previously defined auxiliary constructor or the primary constructor.

We use 'def' keyword to define auxiliary constructors like method/function definition. In Java, constructor definition and Method definition is different.

60. What is the use of 'yield' keyword in Scala's for-comprehension construct?

We can use 'yield' keyword in Scala's for-comprehension construct. 'for/yield' is used to iterate a collection of elements and generates new collection of same type. It does not change the original collection. It generates new collection of same type as original collection type.

For example, if we use 'for/yield' construct to iterate a List then it generates a new List only.

```
scala> val list = List(1,2,3,4,5)
list: List[Int] = List(1, 2, 3, 4, 5)
```

```
scala> for(l <- list) yield l*2
```

```
res0: List[Int] = List(2, 4, 6, 8, 10)
```

61. What is guard in Scala's for-comprehension construct?

In Scala, for-comprehension construct has an if clause which is used to write a condition to filter some elements and generate new collection. This if clause is also known as "Guard".

If that guard is true, then add that element to new collection. Otherwise, it does not add that element to original collection.

Example:- For-comprehension Guard to generate only Even numbers into new collection.

```
scala> val list = List(1,2,3,4,5,6,7,8,9,10)
list: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
scala> for(l <- list if l % 2 == 0) yield l
res0: List[Int] = List(2, 4, 6, 8, 10)
```

62. How Scala solves Inheritance Diamond Problem automatically and easily than Java 8?

If we use Java 8's Interface with Default methods, we will get Inheritance Diamond Problem. Developer has to solve it manually in Java 8. It does not provide default or automatic resolution for this problem.

In Scala, we will get same problem with Traits but Scala is very clever and solves Inheritance Diamond Problem automatically using Class Linearization concept.

63. In Scala, Pattern Matching follows which Design Pattern? In Java, 'instanceof' operator follows which Design Pattern?

In Scala, Pattern Matching follows Visitor Design Pattern. In the same way, Java's 'instanceof' operator also follows Visitor Design Pattern.

=====

64. What is the current latest version of Scala? What is the major change or update in Scala 2.12?

Current Scala's stable is 2.11.7. It supports Java SE 7.

The major change or update in Scala 2.12 version is that it supports Java SE 8 or later versions only. Scala 2.12 is not a binary compatible with the 2.11.x series. It's still in Mile Stone Builds only.

65. What is Option in Scala? What are Some and None? What is Option/Some/None Design Pattern in Scala?

In Scala, Option is used to represent optional values that is either exist or not exist.

Option is an abstract class. Option has two subclasses: Some and None. All three (Option, Some and None) are defined in "scala" package like "scala.Option".

Option is a bounded collection in Scala, which contains either zero or one element. If Option contains zero elements that is None. If Option contains one element, that is Some.

Some is used to represent existing value. None is used to represent non-existent value.

Example:-

```
def get(val index: Int): Option[String]
```

Let us assume that this method is from List. This method has a return type of Option[String]. If List contains elements, this get method returns "Some[String]" element available in that index position. Otherwise, it returns "None" (that is no elements)

Some is a case class and None is an Object. As both are case class/object, we can use them in Pattern Matching very well.

The combination of all these three definitions is known as Option/Some/None Design Pattern in Scala.

66. What is Either in Scala? What are Left and Right in Scala? Explain Either/Left/Right Design Pattern in Scala?

In Scala, Either is an abstract class. It is used to represent one value of two possible types. It takes two type parameters: Either[A,B].

It exactly have two subtypes: Left and Right. If Either[A,B] represents an instance A that means it is Left. If it represents an instance B that means it is Right.

This is known as Either/Left/Right Design Pattern in Scala.

67. What is the equivalent construct of Scala's Option in Java SE 8? What is the use of Option in Scala?

Scala's Option is similar to Java SE 8's Optional. Java SE 8 has introduced a new utility class Optional to represent existing or non-existing of some value. Optional is available in java.util package.

Both Scala's Option and Java SE 8's Optional are used to represent optional values. Both are used to avoid unwanted null checks and NullPointerException.

68. What are the Advantages of Functional Programming (FP) or Advantages of Pure Functions?

The following are the Advantages of Functional Programming (FP) or Advantages of Pure Functions:

More Modular
Easier to understand Or Easier reason about
Easier to test
Less prone to bugs
Easier to reuse
Easier to Parallelism and generalize

69. What are the Popular Scala-Based Frameworks to develop RESTful Web Services or REST API?

There are many Scala-Based Framework to develop RESTful Web Services. Most popular frameworks are:

Play Framework
In Play, we call REST API URLs as routes. We place all routes at once place in Play framework. It is a stateless web framework to develop REST API easily.

Scalatra Framework
It is very simple and easy Scala-based web framework to develop REST API

Spray Framework
It is very concise and built on top of Akka framework so it's better to develop REST API using Actor Model.

Lift Framework
It allows routing using Pattern Matching concept.

70. What is the best Framework to generate REST API documentation for Scala-based applications?

Swagger is is the best tool for this purpose. It is very simple and open-source tool for generating REST APIs documentation with JSON for Scala-based applications.

If we use Play with Scala to develop your REST API, then use play-swagger module for REST API documentation.
If we use Spray with Scala to develop your REST API, then use spray-swagger module for REST API documentation.

71. Like Hibernate for Java-based applications, What are the Popular ORM Frameworks available to use in Play/Scala based applications?

Like JPA, Hibernate and Toplink etc ORM Frameworks for Java-based applications, There are many ORM frameworks to use in Play/Scala based applications.
Popular ORM frameworks for Play/Scala based applications:

Slick
Anorm
SORM(Scala ORM)
Squeryl

72. What is the best tool to develop Play/Scala applications to persist data in MongoDB NoSQL data store?

ReactiveMongo is the best Scala Driver to develop Play/Scala applications to persist data in MongoDB NoSQL data store. It supports fully non-blocking and asynchronous I/O operations.

73. Popular clients who are using Play and Scala to develop their applications?

Thousands of clients are using Play and Scala in Production. The following list is the more popular clients who are using Play and Scala actively.

LinkedIn
The Guardian
Ocado
LuchidChart
GOV.UK

74. What is the best language to use with Play framework: Scala or Java?

Play 2 is completely written in Scala. If we use Java with Play framework, we need to face many issues because Java does not support full FP features.

Scala is the best option to use with Play framework to develop Highly Scalable, Better Performance with Concurrency/Parallelism and Low latency applications, because:

Play 2 is completely written in Scala.

It supports full FP features.

It is more expression language than Java.

It supports Akka Actor model very easily

It supports some new OOP feature like Traits.

Play's built-in templates are developed in Scala

How Scala supports both Highly Scalable and Highly Performance applications?

As Scala supports Multi-Paradigm Programming(Both OOP and FP) and uses Actor Concurrency Model, we can develop very highly Scalable and high-performance applications very easily.

75. What are the available Build Tools to develop Play and Scala based Applications?

The following three are most popular available Build Tools to develop Play and Scala Applications:

SBT

Maven

Gradle

76. What is SBT? What is the best Build Tool to develop Play and Scala Applications?

SBT stands for Scala Build Tool. Its a Simple Build Tool to develop Scala-based applications.

Most of the people uses SBT Build tool for Play and Scala Applications. For example, IntelliJ IDEA Scala Plugin by default uses SBT as Build tool for this purpose.

77. What are the available Unit Testing, Functional Testing and/or BDD Frameworks for Play and Scala Based applications?

The following are most popular available Unit Testing, Functional Testing and/or BDD Frameworks for Play/Scala Based applications:

Spec2

ScalaTest

ScalaCheck

Mokito

78. What is the best Code-coverage tool available for Play and Scala based applications?

SCoverage is the Code-coverage tool for Play and Scala based applications.

SCoverage stands for Scala Code-coverage tool. It has three separate plug-ins to supports the following build tools:

SBT

Maven

Gradle

79. What is the best Scala style checker tool available for Play and Scala based applications?

Like Checkstyle for Java-Based Applications, Scalastyle is best Scala style checker tool available for Play and Scala based applications.

Scalastyle observes our Scala source code and indicates potential problems with it. It has three separate plug-ins to supports the following build tools:

SBT

Maven

Gradle

It has two separate plug-ins to supports the following two IDEs:

IntelliJ IDEA

Eclipse IDE

80. Which IDEs support Play and Scala-Based Applications Development and how?

The following two popular IDEs support Play and Scala-Based Applications Development:

IntelliJ IDEA

Eclipse IDE

They support by using Scala Plugins like Eclipse IDE has a Scala IDE for Eclipse to support Play and Scala-Based Applications Development.

IntelliJ IDEA has a plug-in like “Scala Plugin for IntelliJ IDEA” to support “Scala, SBT and Play 2 Framework” based applications.

What is the default Unit and Functional Testing Framework for Play? What is the default Build Tool for Play? What is the Default Template Engine for Play? What is the built-in Web Server available in Play Framework?

Play Framework’s default Unit and Functional Testing Framework is Spec2. It is very easy to test Play/Scala based applications using Spec2 Framework.

Play Framework’s Default built-in template is “Twirl”. It was developed in Scala. By using these templates, we can develop Play/Scala based applications very easily.

The Built-in or Default Web Server available for Play Framework is Netty Server.

81. Why Scala is better than Java? What are the advantages of Scala over Java (Java 8)? Compare to Java What are the major advantages or benefits of Scala?

Because Scala supports the following extra features, it is better than Java 8:

Full FP Features

More Expression Language

Pattern Matching

Better support for Akka Actor Model

Automatic resolution for Inheritance Diamond Problem with Traits

Asynchronous and Non-blocking IO programming using Akka Framework

Fully Reactive Streaming API

82. What is an Anonymous Function In Scala? What is a Function Literal in Scala? What are the advantages of a Anonymous Function/Function Literal in Scala?

Anonymous Function is also a Function but it does not have any function name. It is also known as a Function Literal.

The advantages of a Anonymous Function/Function Literal in Scala:

We can assign a Function Literal to variable

We can pass a Function Literal to another function/method

We can return a Function Literal as another function/method result/return value.

83. What is an Higher-Order Function (HOF)?

Higher Order Function (HOF) is also a function but which performs one, two or both of the following things:

Take other functions as arguments

Return functions as their results

84. What are the differences between Case class and Normal Class?

Case class is also a class, however when we compare it with normal class, it gives the following extra features or benefits:

By default, Case-class constructor parameters are ‘val’. We don’t need to declare parameters with ‘val’.

By default, Case-class constructor parameters become class fields.

These methods are added automatically: toString, equals, hashCode, copy, apply and unapply.

It automatically gets Companion object.

No need to use ‘new’ keyword to create instance of Case Class.

Easy to use in Pattern Matching.

All these features are added by Scala Compiler at compile-time. It is not possible with normal class.

85. What are the advantages of Play/Scala stack to develop web applications?

The following are the major advantages of Play/Scala stack to develop web applications:

Open Source

Play is an Open-source free-software framework to develop web applications.

Better Productivity

Play framework's Auto-reload feature improves Developer Productivity. No need to build, deploy and test our changes. Just do our changes and refresh the page to see our changes.

Stateless and Easy to Develop REST API

Play is HTTP based stateless model to serve web requests so it is very easy to develop REST API or RESTful Web Services.

Better Error-Handling

If we develop our web application using Play framework, it informs all errors in the browser in very useful format. It shows error message, the file location, line number where error occurred, highlighting the code-snippet to understand the error very easily.

High Performance and Better Scalability With Reactive

Play framework is developed by following Reactive design patterns and it is built on top of Netty server to utilize Non-blocking IO Feature. Because of this feature, we can develop very highly Scalable and performance applications very easily.

Easy to Extend

Play is very flexible framework and supports developing plug-ins very easy to extend its features and functionality.

Highly Concurrency and Better Parallelism

As both Scala and Play supports Functional Programming, it is very easy to develop Highly Concurrency and Better Parallelism applications very easily because FP supports Immutability, Pure Functions (Functions without side-effects), Pattern Matching, Actor Model etc.

Better Reusability, Easy to Test and More Modular

As both Scala and Play supports Functional Programming, we can develop more modular and reusable applications. It is also very easy to test more modular applications.

86. What are the Java's OOP constructs not supported by Scala? What are the Scala's OOP constructs not supported by Java? What are the new OOPs constructs introduced by Scala, but not supported by Java?

Java's OOP constructs, which are not supported by Scala:

There is no interface concept in Scala

There is no Enum concept in Scala

Scala's OOP constructs, which are not supported by Java:

OR

The new OOPs constructs introduced by Scala, but not supported by Java:

Scala Traits

Solving Inheritance Diamond Problem automatically.

87. What is call-by-name? Does Scala and Java support call-by-name? What is the difference between call-by-value and call-by-name function parameters?

Call-by-name means evaluates method/function parameters only when we need them or we access them. If we don't use them, then it does not evaluate them.

Scala supports both call-by-value and call-by-name function parameters. However, Java supports only call-by-value, but not call-by-name.

Difference between call-by-value and call-by-name:

The major difference between these two are described below:

In Call-by-name, the function parameters are evaluated only whenever they are needed but not when the function is called.

In Call-by-value, the function parameters are evaluated when the function is called.

In Call-by-value, the parameters are evaluated before executing function and they are evaluated only once irrespective of how many times we used them in that function.

In Call-by-name, the parameters are evaluated whenever we access them and they are evaluated each time we use them in that function.

Scala Syntax Differences

Call-by-value:

```
def myFunction(a: Int, b: Int) { }
```

Here both a and b are Call-by-value parameters to myFunction.

Call-by-name:

```
def myFunction(a: Int, b: => Int) { }
```

Here both a is a Call-by-value parameter and b is Call-by-name to myFunction.

88. What are the popular MVC frameworks for Scala Language to develop Web Applications?

The following are the most popular MVC frameworks available for Scala Language to develop Web Applications:

Play Framework
Scalatra Framework
Spray Framework
Lift Framework

89. What are major differences between Java-Based and Scala-Based Maven Project's structure?

Most of the Java-based projects uses Maven as their Build tool. However, most of the people uses SBT as build tool to develop their Scala-based applications, but Some Teams uses Maven too as a build tool to develop their Scala-based applications.

Maven folder structure for both Java-based and Scala-based projects is almost same just one folder name change as shown in the below diagrams: java and scala folder names.

Java-based Maven Project Folder Structure:

java_maven_project_structure

Scala-based Maven Project Folder Structure:

scala_maven_project_structure

90. What is Extractor in Scala? What is the difference between Constructor and Extractor in Scala? What is the use of Extractor in Scala?

Not only in Java and Scala, in almost all OOP languages Constructor is used to create (or assemble) an object or an instance of a Class using it's parameters (or components). Extractor is quite opposite to Constructor.

In Scala, Extractor is used to decompose or disassemble an object into it's parameters (or components).

In Scala, apply method is a Constructor. Internally, Extractor uses unapply method to decompose an objects into it's parts (or parameters). In Scala, Extractor is mainly used in Pattern Matching concept. We will discuss Pattern Matching concept soon.

91. What is the use of '???' in Scala-based Applications?

This '???' three question marks is not an operator, a method in Scala. It is used to mark a method which is 'In Progress' that means Developer should provide implementation for that one.

This method is define in scala.PreDef class as shown below:

```
def ??? : Nothing = throw new NotImplementedError
```

If we run that method without providing implementation, then it throws 'NotImplementedError' error as shown below:

```
scala> def add(a:Int, b:Int) : Int = ???  
add: (a: Int, b: Int)Int
```

```
scala> add(10,20)  
scala.NotImplementedError: an implementation is missing
```

92. Explain the main difference between List and Stream in Scala Collection API? How do we prove that difference? When do we choose Stream?

In Scala, both List and Stream are from Collection API and works almost similar. Both are Immutable collections.

However, there is one main difference between List and Stream in Scala Collection API: That is List elements are evaluated Eagerly and Stream elements are evaluated Lazily that means when we access them.

```
scala> var list1 = List(1,2,3,4)
list1: List[Int] = List(1, 2, 3, 4)
```

Here we can observe that all List elements evaluated at the time of creating List object. However, if we do same thing on Stream, we cannot see all elements. We can see only first evaluated element and remaining elements are evaluated lazily as shown below:

```
scala> var s1 = Stream(1,2,3,4)
s1: scala.collection.immutable.Stream[Int] = Stream(1, ?)
```

When we want Lazy collection to evaluate elements only when we access them then it's better to use Stream.

93. What is the difference between :: and #:: in Scala? What is the difference between ::: and #::: in Scala?

In Scala Collection API,

:: and ::: are methods available in List class.

#:: and #::: are methods available in Stream class

In List class, :: method is used to append an element to the beginning of the list.

```
scala> var list1 = List(1,2,3,4)
list1: List[Int] = List(1, 2, 3, 4)
```

```
scala> list1 = 0 :: list1
list1: List[Int] = List(0, 1, 2, 3, 4)
```

In List class, ::: method is used to concatenate the elements of a given list in front of this list.

```
scala> var list1 = List(3,4,5)
list1: List[Int] = List(3, 4, 5)
```

```
scala> val list2 = List(1,2) ::: list1
list2: List[Int] = List(1, 2, 0, 1, 2, 3, 4)
```

In Stream class, #:: method is used to append a given element at beginning of the stream. Only this newly added element is evaluated and followed by lazily evaluated stream elements.

```
scala> var s1 = Stream(1,2,3,4)
s1: scala.collection.immutable.Stream[Int] = Stream(1, ?)
```

```
scala> s1 = 0 #:: s1
s1: scala.collection.immutable.Stream[Int] = Stream(0, ?)
```

In Stream class, #::: method is used to concatenate a given stream at beginning of the stream. Only this newly added element is evaluated and followed by lazily evaluated stream elements.

```
scala> var s1 = Stream(1,2,3,4)
s1: scala.collection.immutable.Stream[Int] = Stream(1, ?)
```

```
scala> val s2 = Stream(-1,0) #::: s1
s2: scala.collection.immutable.Stream[Int] = Stream(-1, ?)
```

:: method works as a cons operator for List class and #:: method works as a cons operator for Stream class. Here 'cons' stands for construct.

::: method works as a concatenation operator for List class and #::: method works as a concatenation operator for Stream class.

94. If I want to become a Fullstack Scala Developer, which technology stack I should learn?

If you want to become a Fullstack Scala Developer, you should learn the following technology stack:

Scala 2.11.7
Play 2.4.6 Framework
Akka 2.3 Framework
One Build Tool: SBT/Maven
One JS Framework: CoffeeScript/JavaScript
One IDE: IntelliJ IDEA 15/ Eclipse IDE 4.x
One TDD & BDD Framework: ScalaTest, Spec2, ScalaCheck, Mockito
Micro Services with Play and Scala
SCoverage
Scalastyle
Functional Programming Design Patterns
Machine Learning with Scala

NOTE:- In Scala, Extractor follows Extractor Design Pattern. If you want to learn it in depth, please go through my Scala Tutorial (Most of the posts follow this pattern: Scala xxxx In Depth where xxxx is a concept like Extractor).

=====

Language Questions:

What is the difference between a var, a val and def?
What is the difference between a trait and an abstract class?
What is the difference between an object and a class?
What is a case class?
What is the difference between a Java future and a Scala future?
What is the difference between unapply and apply, when would you use them?
What is a companion object?
What is the difference between the following terms and types in Scala: Nil, Null, None, Nothing?
What is Unit?
What is the difference between a call-by-value and call-by-name parameter?
How does Scala's Stream trait leverage call-by-name?
Define uses for the Option monad and good practices it provides.
How does yield work?
Explain the implicit parameter precedence.
What operations is a for comprehension syntactic sugar for?
Streams:
What consideration you need to have when you use Scala's Streams?
What technique does the Scala's Streams use internally?
Functional Programming Questions:

What is a functor?
What is a applicative?
What is a monad?
What are the monad axioms?
What Scala data types are, or behave like, monads?
What are the basic and optional requirement/s to conform a Monad?
Explain higher order functions.
What is gained from using immutable objects?
What is tail recursion?
How does it differentiate from common recursion?
What issues are there with tail recursive functions in the JVM?
How does the Scala compiler optimize a tail recursive function?
How do you ensure that the compiler optimizes the tail recursive function?
What is function currying?
What are implicit parameters?
What are typeclasses?
What are lenses?
What is and which are the uses of: Enumerators, Enumerators and Iteratee
Reactive Programming Questions:

Explain the actor model.
What are benefits of non-blocking (asynchronous I/O) over blocking (synchronous I/O).
Do you think that Scala has the same async spirit as Node.js?
Explain the difference between concurrency and parallelism, and name some constructs you can use in Scala to leverage both.
What is the global ExecutionContext?
What does the global ExecutionContext underlay?

Akka:
Which are the 3 main components in a Stream?

Coding Questions:

How can you make a List[String] from a List[List[String]]?
What is the difference (if any) between these 2 statements?
var x = immutable.Set[Int]()
val y = mutable.Set[Int]()

=====

1) Explain what is Scala?

Scala is an object functional programming and scripting language for general software applications designed to express solutions in a concise manner.

2) What is a 'Scala set'? What are methods through which operation sets are expressed?

Scala set is a collection of pairwise elements of the same type. Scala set does not contain any duplicate elements. There are two kinds of sets, mutable and immutable.

3) What is a 'Scala map'?

Scala map is a collection of key or value pairs. Based on its key any value can be retrieved. Values are not unique but keys are unique in the Map.

4) What is the advantage of Scala?

- a) Less error prone functional style
- b) High maintainability and productivity
- c) High scalability
- d) High testability
- e) Provides features of concurrent programming

5) In what ways Scala is better than other programming language?

- a) The arrays uses regular generics, while in other language, generics are bolted on as an afterthought and are completely separate but have overlapping behaviours with arrays.
- b) Scala has immutable "val" as a first class language feature. The "val" of scala is similar to Java final variables. Contents may mutate but top reference is immutable.
- c) Scala lets 'if blocks', 'for-yield loops', and 'code' in braces to return a value. It is more preferable, and eliminates the need for a separate ternary operator.
- d) Singleton has singleton objects rather than C++/Java/ C# classic static. It is a cleaner solution
- e) Persistent immutable collections are the default and built into the standard library.
- f) It has native tuples and a concise code
- g) It has no boiler plate code

6) What are the Scala variables?

Values and variables are two shapes that come in Scala. A value variable is constant and cannot be changed once assigned. It is immutable, while a regular variable, on the other hand, is mutable, and you can change the value.

The two types of variables are

```
var myVar : Int=0;  
val myVal: Int=1;
```

7) Mention the difference between an object and a class ?

A class is a definition for a description. It defines a type in terms of methods and composition of other types. A class is a blueprint of the object. While, an object is a singleton, an instance of a class which is unique. An anonymous class is created for every object in the code, it inherits from whatever classes you declared object to implement.

8) What is recursion tail in scala?

'Recursion' is a function that calls itself. A function that calls itself, for example, a function 'A' calls function 'B', which calls the function 'C'. It is a technique used frequently in functional programming. In order for a tail recursive, the call back to the function must be the last function to be performed.

9) What is 'scala trait' in scala?

'Traits' are used to define object types specified by the signature of the supported methods. Scala allows to be partially implemented but traits may not have constructor parameters. A trait consists of method and field definition, by mixing them into classes it can be reused.

10) When can you use traits?

There is no specific rule when you can use traits, but there is a guideline which you can

consider.

- a) If the behaviour will not be reused, then make it a concrete class. Anyhow it is not a reusable behaviour.
- b) In order to inherit from it in Java code, an abstract class can be used.
- c) If efficiency is a priority then lean towards using a class
- d) Make it a trait if it might be reused in multiple and unrelated classes. In different parts of the class hierarchy only traits can be mixed into different parts.
- e) You can use abstract class, if you want to distribute it in compiled form and expects outside groups to write classes inheriting from it.

11) What is Case Classes?

Case classes provides a recursive decomposition mechanism via pattern matching, it is a regular classes which export their constructor parameter. The constructor parameters of case classes can be accessed directly and are treated as public values.

12) What is the use of tuples in scala?

Scala tuples combine a fixed number of items together so that they can be passed around as whole. A tuple is immutable and can hold objects with different types, unlike an array or list.

13) What is function currying in Scala?

Currying is the technique of transforming a function that takes multiple arguments into a function that takes a single argument Many of the same techniques as language like Haskell and LISP are supported by Scala. Function currying is one of the least used and misunderstood one.

14) What are implicit parameters in Scala?

Implicit parameter is the way that allows parameters of a method to be “found”. It is similar to default parameters, but it has a different mechanism for finding the “default” value. The implicit parameter is a parameter to method or constructor that is marked as implicit. This means if a parameter value is not mentioned then the compiler will search for an “implicit” value defined within a scope.

15) What is a closure in Scala?

A closure is a function whose return value depends on the value of the variables declared outside the function.

16) What is Monad in Scala?

A monad is an object that wraps another object. You pass the Monad mini-programs, i.e functions, to perform the data manipulation of the underlying object, instead of manipulating the object directly. Monad chooses how to apply the program to the underlying object.

17) What is Scala anonymous function?

In a source code, anonymous functions are called ‘function literals’ and at run time, function literals are instantiated into objects called function values. Scala provides a relatively easy syntax for defining anonymous functions.

18) Explain ‘Scala higher order’ functions?

Scala allows the definition of higher order functions. These are functions that take other functions as parameters, or whose result is a function. In the following example, apply () function takes another function ‘f’ and a value ‘v’ and applies function to v.

Example:

```
[crayon-583155ef4894e105317284/]
```

When the above code is compiled and executed, it produces following result.

```
[crayon-583155ef48954189515863/]
```

19) What is the difference between var and value?

In scala, you can define a variable using either a, val or var keywords. The difference between val and var is, var is much like java declaration, but val is little different. We cannot change the reference to point to another reference, once the variable is declared using val. The variable defined using var keywords are mutable and can be changed any number of times.

20) What are option, some and none in scala?

‘Option’ is a Scala generic type that can either be ‘some’ generic value or none. ‘Queue’ often uses it to represent primitives that may be null.

21) How do I append to the list?

In scala to append into a list, use “:++” single value

```
[crayon-583155ef48957217370776/]
```

22) How can you format a string?

To format a string, use the .format () method, in scala you can use
Val formatted= “%s %i”.format (mystring.myInt)

23) Why scala prefers immutability?

Scala prefers immutability in design and in many cases uses it as default. Immutability can help when dealing with equality issues or concurrent programs.

24) What are the four types of scala identifiers ?

The four types of identifiers are

- a) Alpha numeric identifiers
- b) Operator identifiers
- c) Mixed identifiers
- d) Literal identifiers

25) What are the different types of Scala literals?

The different types of literals in scala are

- a) Integer literals
- b) Floating point literals
- c) Boolean literals
- d) Symbol literals
- e) Character literals
- f) String literals
- g) Multi-Line strings