* 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 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.

**Is Scala Statically-Typed Language? What is Statically- and 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.
* 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.

**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, Implicit etc.

**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, Implicit etc.

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

* 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
* Highly Productivity
* Distributed Applications

**Drawbacks of Scala Language:-**

* Less Readable Code
* Bit tough to Understand the Code for beginners
* Complex Syntax to learn
* Less Backward Compatibility

### 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”.

### 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.

### 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”.

### 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

### 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.

### 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’.

### 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

### 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)**“.

### 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

### 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.

### 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.

### 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.

### 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 Langauge. It is very tough to deal static members in Concurrency applications.

h3>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.

That’s it all about “Scala Basic Interview Questions and Answers”. We will discuss some Intermediate, Advanced and Real-time Scala Interview Questions and Answers in my coming posts.