1. Best Scala Interview Questions

In this best 30 Scala Interview Questions, we are going to cover all the frequently asked questions in Scala Interview. These Scala Interview Questions are provided by **[Scala](https://data-flair.training/blogs/scala-tutorial/)** experts which are beneficial for both freshers as well as experienced. You can gain a better understanding of Scala with these Scala Interview Questions and answers. Follow each link, to gain in-depth knowledge of Scala.  
So let’s start Scala Interview Questions.

2. Scala Interview Questions and Answers

Below, we are discussing some important Scala Interview Questions for both freshers and experienced:

**Q.1. What is ofDim in Scala?**

ofDim() is a method in Scala that lets us create multidimensional arrays. Since these let us store data in more than one dimension, we can store data like in a matrix. Let’s take an example.

1. scala> import Array.ofDim
2. import Array.ofDim
3. scala> var a=ofDim[Int](3,3)
4. a: Array[Array[Int]] = **Array**(**Array**(0, 0, 0), **Array**(0, 0, 0), **Array**(0, 0, 0))
5. scala> var k=1
6. k: Int = 1
7. scala> **for**(i<-0 to 2){
8. | **for**(j<-0 to 2){
9. | **a**(i)(j)={i+k}
10. | k+=1
11. | }
12. | k-=1
13. | }
14. scala> a

**res12:** Array[Array[Int]] = Array(Array(1, 2, 3), Array(4, 5, 6), Array(7, 8, 9)).

**Q.2. What do you have to say about exception propagation in Scala?**

When a function experiences an exception, it looks for a handler to deal with it. When it fails to find one, it searches for one in the caller method. Failing there, it looks for yet another in the next caller in the chain. Whenever it does find a handler, it makes it catch the exception. This is exception propagation.

[**Read Scala Functions**](https://data-flair.training/blogs/scala-functions/)

**Q.3. What is a BitSet?**

A bitset is a set of non-negative integers depicted as arrays. These arrays are variable in size and packed into 64-bit words. The largest number in a bitset determines its memory footprint. Let’s take an example.

1. scala> import scala.collection.immutable.\_
2. import scala.collection.immutable.\_
3. scala> var nums=**BitSet**(7,2,4,3,1)
4. nums: scala.collection.immutable.BitSet = **BitSet**(1, 2, 3, 4, 7)
5. scala> nums+=9 //Adding an element
6. scala> nums

res14: scala.collection.immutable.BitSet = BitSet(1, 2, 3, 4, 7, 9)

1. scala> nums-=4 //Deleting an element
2. scala> nums

res16: scala.collection.immutable.BitSet = BitSet(1, 2, 3, 7, 9)

1. scala> nums-=0 //Deleting an element that doesn’t exist
2. scala> nums

res18: scala.collection.immutable.BitSet = BitSet(1, 2, 3, 7, 9)

**Q.4. What is a vector in Scala?**

A vector is a general-purpose data structure that is immutable. We can use it when we want to hold a huge number of elements and want random access to them. This data structure extends the trait IndexedSeq and the abstract class AbstractSeq.

1. scala> import scala.collection.immutable.\_
2. import scala.collection.immutable.\_
3. scala> var v1=Vector.empty
4. v1: scala.collection.immutable.Vector[Nothing] = **Vector**()
5. scala> var v2=**Vector**(7,2,4,3,1)
6. v2: scala.collection.immutable.Vector[Int] = **Vector**(7, 2, 4, 3, 1)
7. scala> var v3:Vector[Int]=**Vector**(8,2,6,5,9)
8. v3: scala.collection.immutable.Vector[Int] = **Vector**(8, 2, 6, 5, 9)
9. scala> v3=v3 :+7 //Adding a new element
10. v3: scala.collection.immutable.Vector[Int] = **Vector**(8, 2, 6, 5, 9, 7)
11. scala> v2++v3 //Merging two vectors

res19: scala.collection.immutable.Vector[Int] = Vector(7, 2, 4, 3, 1, 8, 2, 6, 5, 9, 7)

1. scala> v3.reverse //Reversing a vector

res20: scala.collection.immutable.Vector[Int] = Vector(7, 9, 5, 6, 2, 8)

1. scala> v3.sorted //Sorting a vector

res21: scala.collection.immutable.Vector[Int] = Vector(2, 5, 6, 7, 8, 9)

In results 20 and 21, we do not assign the expression to any variable, so not that this doesn’t change the original vectors.

**Q.5. Explain streams in Scala.**

A stream is a lazy list as it evaluates elements only when it needs to. This lazy computation enhances program performance.

1. scala> val stream=177#::199#::69#::Stream.empty
2. stream: scala.collection.immutable.Stream[Int] = **Stream**(177, ?)

Since we don’t need the second element yet, Scala doesn’t evaluate it.

1. scala> val stream1=(1 to 7).toStream
2. stream1: scala.collection.immutable.Stream[Int] = **Stream**(1, ?)
3. scala> stream.head

res22: Int = 177

1. scala> stream.map{\_\*2}

res24: scala.collection.immutable.Stream[Int] = Stream(354, ?)

[**Learn Scala Arrays and Multidimensional Arrays in Scala**](https://data-flair.training/blogs/scala-array/)

**Q.6. What are the advantages of Scala?**

Among various other benefits of the language, here are a few:

* It is highly scalable
* It is highly testable
* It is highly maintainable and productive
* It facilitates concurrent programming
* It is both object-oriented and functional
* It has no boilerplate code
* Singleton objects are a cleaner solution than static
* Scala arrays use regular generics
* Scala has native tuples and concise code

For a detailed piece on its benefits, read up on[**Advantages of Scala**](https://data-flair.training/blogs/scala-advantages/).

**Q.7. Who designed Scala? Which is the latest version?**

At the time of writing, Scala 2.12.6 is the latest version. The interviewer may ask you this to find out whether you keep yourself updated. Martin Odersky, a German computer scientist, began designing it in 2001 at EPFL, Switzerland.

**Q.8. How is *Val* different from *var* in Scala?**

In this language, *val* is a value and *var* is a variable. These are two different keywords for declaring immutable and mutable entities respectively. This means that you can always reassign a *var*, but trying to do that to a *val* makes the compiler throw an error.

1. scala> val c=7
2. c: Int = 7
3. scala> c=8
4. <console>:19: error: reassignment to val
5. c=8
6. ^
7. scala> var c=7
8. c: Int = 7
9. scala> c=8
10. c: Int = 8

**Q.9. Why do we need App in Scala?**

App is a helper class that holds the main method. We can have our classes extend App to render executable code:

1. scala> object Hello extends App{
2. | **println**("Hello")
3. | }
4. defined object Hello

With this code, our object Hello inherits the main method from the App trait.

**Q.10. How is a class different from an object?**

A class is a blueprint, a definition. In terms of methods and compositions of other types, it defines a type. An object, however, is a singleton. It is a unique instance of a class. Every object in your code has an anonymous class for it. Where in Java, you would use a class with static members, you use an object in Scala.

Read more on [**Classes and Objects in Scala**](https://data-flair.training/blogs/scala-object-oriented-programming/)**.**

**Scala Interview Questions for Freshers – Q. 1,2,3,4,5,6,7,10**

**Scala Interview Questions for Experienced – Q. 8,9**

**Q.11. How do the terms ‘Null’, ‘Nil’, ‘None’, and ‘Nothing’ differ in Scala?**

While they appear similar, the mentioned terms are slightly different in their behaviors. Here’s how:

* Null represents the absence of value. It depicts the absence of type information for complex types inherited from AnyRef.
* Nil denotes the end a List.
* None is the value of an Option with no value in it.
* Nothing is the lowest type in the entire type system. All values under AnyVal and AnyRef fall under this. A method that throws an exception uses Nothing as a return type.

**Q.12. What is a monad in Scala?**

A monad is something to which we can pass functions and manipulate the underlying object’s data. We don’t need to manipulate the object directly. Hence, a monad is an object that wraps another.

**Q.13. Differentiate a Scala function from a Java method.**

In Scala, a function is also a value. Unlike in Java, we can assign it to vals and vars, and also return it from another function. Check [**Higher-Order Functions in Scala**](https://data-flair.training/blogs/scala-functions/). Since Java 8, we can use lambda expressions to use functions as first-class objects. So, we can pass functions to methods.

**Q.14. Explain vararg arguments.**

With varargs, we can pass a variable number of arguments to a method.

1. scala> def **func**(arg:String\*)=arg.**mkString**(", ")
2. func: (arg: String\*)String
3. scala> **func**("red","green","blue")

res28: String = red, green, blue

**Q.15. What do you know about traits in Scala?**

A trait is like a partially implemented interface that can hold abstract and non-abstract methods. They’re like Java interfaces; that is what Scala compiles them into. Let’s take an example.

[**Learn more about Scala trait**](https://data-flair.training/blogs/scala-trait/)

1. scala> trait Hello{
2. | def **sayhello**()
3. | }
4. defined trait Hello

**warning:** previously defined object Hello is not a companion to trait Hello.  
Companions must be defined together; you may wish to use: paste mode for this.

1. scala> class A extends Hello{
2. | def **sayhello**(){
3. | **println**("Hello")
4. | }
5. | }
6. defined class A
7. scala> var a=new **A**()
8. a: A = A@10e595ed
9. scala> a.**sayhello**()
10. Hello

**Q.16. What is an Option in Scala?**  
A **[Scala Option](https://data-flair.training/blogs/scala-option/)** is a kind of a container. It can hold zero or one element of a type. When it holds a value, it holds Some[T]; otherwise, it holds a None object, as we discussed in the previous question.

1. scala> val o:Option[Int]=**Some**(7)
2. o: Option[Int] = **Some**(7)
3. scala> val o1:Option[Int]=None
4. o1: Option[Int] = None

**Q.17. Is a case class the same as a regular class in Scala?**  
No, these aren’t synonyms. Here are a few important characteristics of a **[Scala case class](https://data-flair.training/blogs/scala-case-class/)**:

* They support pattern-matching
* To create an instance of a case class, you don’t need *new*
* Scala automatically generates methods like equals(), hashcode(), and toString() for case classes
* For all constructor arguments for a case class, Scala automatically generates accessor methods

For more on case classes, read up on Case Classes in Scala.  
**Q.18. What is tail-recursion in Scala?**  
Recursion is when a function makes a call to itself. When we place this call as the last action performed in the function, we can call the function *tail-recursive*.

1. scala> def **factorial**(n:Int):Int={
2. | **if**(n==1) return 1
3. | n\***factorial**(n-1)
4. | }
5. factorial: (n: Int)Int
6. scala> **factorial**(5)

res30: Int = 120  
**Q.19. What is a higher-order function in Scala?**  
This is a feature of Scala. A higher-order function is one that takes another as a parameter, or that returns a function.

1. scala> def **func1**(s:String){
2. | **println**("I love "+s)
3. | }
4. func1: (s: String)Unit
5. scala> def **func2**(f:String=>Unit,s:String){
6. | **f**(s)
7. | }
8. func2: (f: String => Unit, s: String)Unit
9. scala> **func2**(func1, "pizza")
10. I love pizza

**Q.20 Explain the working of yield in Scala.**

Used with a loop, *yield* produces a value for each iteration. Another way to do is to use map/flatMap and filter with nomads.

1. scala> **for**(i<-1 to 4) yield i\*3

res35: scala.collection.immutable.IndexedSeq[Int] = Vector(3, 6, 9, 12)

**Scala Interview Questions for Freshers – Q. 12,13,14,15,16,17,18,19**

**Scala Interview Questions for Experienced – Q. 11,20**

**Q.21. Prove that Scala is a language statically/strongly typed.**

Since the compiler performs type checking at compile time instead of runtime, it lets the developer notice and resolve errors at the compile time itself. Hence, we can say that it is strongly and statically typed. Read up on type inference in Features of Scala.

[**Learn Scala Map**](https://data-flair.training/blogs/scala-map/)

**Q.22. How do you use Scala to append to a List?**

For this purpose, we use the single value ‘:+’.

1. scala> var a=List.empty[String]
2. a: List[String] = **List**()
3. scala> a :+="red"
4. scala> a :+="green"
5. scala> a :+="blue"
6. scala> a

res40: List[String] = List(red, green, blue)

1. //And now, appending a List to this
2. scala> a++=**List**("golden","bronze")
3. scala> a

res42: List[String] = List(red, green, blue, golden, bronze)

[**Learn about Scala List**](https://data-flair.training/blogs/scala-list/)

**Q.23. Are *concurrency* and *parallelism* the same thing? Explain.**

When we take a task and break it into subtasks to execute at one time by multiple threads, we call it parallelism. Concurrency, however, is when multiple computations execute sequentially; this is during overlapping time periods. When we avoid access to a mutable state by multiple threads at a time, it is concurrency. Sometimes, actors can concurrent as well as parallel. Node.js is a single-threaded implementation yet is concurrent because of its event loop. An example of parallelism is parallel collections.

**Q.24. Explain different types of identifiers in Scala.**

We have four kinds of identifiers in Scala.

**a. Alphanumeric Identifiers**

These contain letters, underscores, and digits, but only begin with a letter or with an underscore. We name them in camel case. Here are a few examples: ab12, myVal, Pi.

**b. Operator Identifiers**

These contain operator characters except these- ( ) [ ] { } ‘ ” \_ . , ; , `. Some valid examples are: +  => <?> ::: .

**c. Mixed Identifiers**

These contain an alphanumeric identifier, an underscore, and also an operator identifier. Here are some valid examples: myVar\_=, unary\_+.

**d. Literal Identifiers**

These contain an arbitrary string enclosed in backticks(`). Some valid examples are: `class`, `Hello, World!`.

**Q.25. Is Scala compatible with Java? Explain.**

We’ve seen that both Scala and[**Java**](https://data-flair.training/blogs/java-tutorial/) work on the[**JVM**](https://data-flair.training/blogs/java-virtual-machine-jvm/) on the backend. Well, **[Scala classes](https://data-flair.training/blogs/scala-case-class/)** are Java classes and[**Java classes**](https://data-flair.training/blogs/class-and-object-in-java/) are Scala classes too. So you can call a [**Java method**](https://data-flair.training/blogs/java-character-class/) from a **[Scala method](https://data-flair.training/blogs/scala-string-method/)**, and vice-versa. You can also extend classes from one language in another. However, features like [**traits in Scala**](https://data-flair.training/blogs/scala-trait/) have no equivalents in Java.

[**Have a look at Scala vs Java**](https://data-flair.training/blogs/scala-vs-java-performance/)

**Q.26. What is a closure in Scala?**

A[**closure in Scala**](https://data-flair.training/blogs/scala-closures/) is a function whose value depends on variables declared outside of it. Let’s take an example.

1. scala> var c=5
2. c: Int = 5
3. scala> val mul2=(a:Int,b:Int)=>(a+b)\*c
4. mul2: (Int, Int) => Int = $$Lambda$1533/239864031@6d90e705
5. scala> **mul2**(2,3)

res45: Int = 25

1. scala> c=7
2. c: Int = 7
3. scala> **mul2**(2,3)

res46: Int = 35

In this example, mul2 reads the new value of ‘c’ when we call it a second time.

For more on closures, refer to Closures in Scala.

**Q.27. Explain implicit parameter precedence.**

The compiler doesn’t randomly look for implicits in your code; it follows the following precedence:

* Locally declared implicits
* Imported implicits
* Outer scope (ex- a class for a method)
* Inheritance
* Package object
* Implicit scope like companion objects

**Q.28. What is a lens in Scala?**

A lens is an abstraction from functional programming. It makes updating complex immutable nested objects easier for us.

For lenses, we have three kinds of available implementations:

* scalaz.Lens
* Quicklens- Has more functionality than a Sauron
* Sauron

**Q.29. Consider for-comprehensions in Scala. What are they syntactic sugars for?**

A for-comprehension is one way to carry out the composition of operations on monads. We can replace it with a *foreach* or a *map/flatMap* and *filter*.

**Q.30. What is function currying in Scala?**

With[**Scala currying**](https://data-flair.training/blogs/scala-currying/), we can take a function that takes multiple arguments and turn it into a series of functions that take single arguments each. These come in handy working with higher-order functions. With this, we can also fill in only the arguments we have yet.

1. scala> def **mul**(a:Int,b:Int)=a\*b
2. mul: (a: Int, b: Int)Int
3. scala> **mul**(3,4)

res48: Int = 12

We can define it as follows:

1. scala> def **mul**(a:Int)(b:Int)=a\*b
2. mul: (a: Int)(b: Int)Int
3. scala> val mid=**mul**(3)(\_)
4. mid: Int => Int = $$Lambda$1540/90644757@43fe3f7
5. scala> **mid**(4)

res49: Int = 12

**Scala Interview Questions for Freshers – Q. 21,23,24,25,26,27,28,30**

**Scala Interview Questions for Experienced – Q. 29,22**

So, this was all about Scala Interview Questions. Hope it helps.

Top Scala Interview Questions and Answers

Below, we are discussing some frequently asked Scala Interview Questions and Answers:

**Q.1. What is type inference in Scala?**

The Scala compiler decides data types or result types of elements like variables, expressions, and objects at compile-time.   
Scala infers the type for an expression fully or partially. This makes it work even when we don’t declare it explicitly. It also allows for type checking without type annotations. The compiler considers types of atomic values or subexpressions and aggregates it to decide a type for the whole expression.

[**For more on this, read up on Features of Scala.**](https://goo.gl/lZJOWF)

**Q.2. Is Int in Scala the same as java.lang.Integer in Java?**

While the two are similar in some ways, they also differ in other ways:

**Differences:**

* Int does not implement the Comparable interface; java.lang.Integer does.

**Similarities:**

* They’re both classes
* They are 32-bit signed integers
* They represent integer numbers

**Q.3. What is an anonymous function?**

An anonymous [**function in Scala**](https://data-flair.training/blogs/partial-functions-scala-guide/) is a function literal. At runtime, Scala instantiates into a function value. Since this is a way to create a function using just one line of code, we can call it lightweight. Here’s an example:

1. scala> val mul=(a:Int,b:Int)=>a\*b
2. mul: (Int, Int) => Int = $$Lambda$1541/400146874@3aca06a3
3. scala> **mul**(3,4)

res50: Int = 12

**Q.4. Explain implicit parameters.**

Much like default parameters, implicit parameters help us find default values. To make a parameter implicit, we can pass it to a method or a constructor and label it as *implicit*. So when we don’t mention the value of a parameter, the compiler searches for an implicit value defined within a scope.

In Part I, we discussed the precedence for implicit parameters.

**Q.5. What is a case class?**

[**Scala Case classes**](https://data-flair.training/blogs/scala-case-class/) let us model immutable data. For object construction, these classes have an apply() method to handle object construction. These classes have all *val*s, and so prove useful with pattern-matching. Let’s take an example.

1. scala> case class **Book**(title:String, author:String, ISBN: String)
2. defined class Book

**Q.6. What can you tell me about the Scala REPL?**

While we often call it ripple, REPL expands to Read Evaluate Print Loop. We can access it from the command prompt by typing ‘scala’:

1. C:\Users\lifei>scala

Welcome to Scala 2.12.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0\_161).

Type in expressions for evaluation. Or try :help.  
scala>

This is the Command Line Interface(CLI) and command-line shell for Scala, and we can execute Scala code in it as if in an interpreter. We can build and test small pieces of code in it.

**Q.7. What is a Map in Scala?**

A map is a collection in Scala that holds key-value pairs. A map can only hold unique keys, but values can be repeated. We also call these hash tables.

By default, maps are immutable, but we can use the scala.collection.mutable.Map class for a mutable map. This is an example:

1. scala> var m=**Map**("red"->0,"green"->1,"blue"->2)
2. m: scala.collection.immutable.Map[String,Int] = **Map**(red -> 0, green -> 1, blue -> 2)

[**For more on maps, read up on Maps in Scala.**](https://data-flair.training/blogs/scala-map/)

**Q.8. Differentiate between arrays and lists in Scala.**

1. scala> var m=**Map**("red"->0,"green"->1,"blue"->2)

We have the following differences:

* [**Arrays**](https://data-flair.training/blogs/scala-array/) are mutable; lists are immutable
* Arrays are invariants; lists are covariants
* In size, arrays are fixed, but lists are variable
* We can modify array values later, but that is not the same with lists

**Q.9. Is Scala a pure object-oriented language?**

Let’s see the [**object-oriented language.**](https://data-flair.training/blogs/scala-object-oriented-programming/)

* There are no static members
* Scala treats functions and primitives as objects

So, we can conclude that Scala is indeed purely object-oriented as a language.

**Q.10. What is a lazy val?**

Scala executes a val when we define it, but executes a lazy val only when we access it the first time. We declare it using the keyword *lazy*:

1. scala> lazy val x=7
2. x: Int = <lazy>
3. scala> lazy val x=2+3
4. x: Int = <lazy>

Where eager evaluation is at compile-time, lazy evaluation is at run-time.\

**Scala Interview Questions and Answers for freshers- Q. 1,3,4,5,7,8,9,10**

**Scala Interview Questions and Answers for experienced- Q. 2,6**

**Q.11. How many reserved keywords of Scala can you recall?**

Scala has 39 keywords-

abstract, case, catch, class, def, do, else, extends, false, final, finally, for, forSome, if, implicit, import, lazy, match, new, null, object, override, package, private, protected, return, sealed, super, this, throw, trait, try, true, type, val, var, while, with and, yield.

[**Let’s learn Scala Throw Keyword**](https://data-flair.training/blogs/scala-throw-keyword/)

**Q.12. Is it possible to use a Scala keyword as an identifier?**

While it might instinctively seem otherwise, you can use a reserved keyword as an identifier in Scala. Use backticks for this. As an example, we can use backticks on the word *yield* to use Java’s *yield* method from Thread class instead of Scala’s yield.  
Thread.`yield`()

**Q.13. Is an expression the same as a statement in a Scala?**

A statement is one or more operation, but an expression is a value. This means we can assign an expression to a variable, but we can’t assign a statement to the same. Examples of expressions- Scala’s if-condition, Java’s ternary operator. Examples of statements- Java’s if-condition.

[**Let’s revise Scala Regular expression**](https://data-flair.training/blogs/scala-regex/)

**Q.14. Differentiate between equals() and == in Scala. Is the latter the same as == in Java?**

When we try to compare two instances with the == operator, Scala calls the object’s equals() method. We use it to check instance equality. However, in Java, we use it to check reference equality. This tells us whether two references point to the same object.

**Q.15. Explain the differences between the if-else statements for Scala and Java.**

There is one main difference. This statement is an expression in Scala, but not in Java. There, we cannot assign it to a variable because it doesn’t return a value.

1. Scala:
2. scala> val x=3
3. x: Int = 3
4. scala> val kind= **if**(x%2==0) "even" else "odd"
5. kind: String = odd
6. Java:
7. int x=3;
8. String kind;
9. **if**(x%2==0)
10. kind=”even”;
11. else
12. kind=”odd”;

[**Learn more about Scala If-else Statements**](https://data-flair.training/blogs/scala-else-statements-statements/)

**Q.16. How are Scala’s inner classes different from those of Java?**

In Scala, an inner class is associated with the outer class’ object. In Java, however, an inner class is associated with the outer class; the inner class is a member of the outer class.

**Q.17. Is Scala expression-based or statement-based? What about Java?**

Yes, Scala is expression-based. You already know everything is a value in Scala. This means all expressions and statements evaluate to a value. So, we can assign expressions, functions, closures, objects, and other entities to variables. But things aren’t the same with Java. Hence, Java is statement-based.

**Q.18. Why doesn’t Scala have the keyword ‘static’?**

The team made this decision to maintain the nature of Scala as a pure object-oriented language. If they included it, we could access class members without having to create an object. That would violate the principles of OOP. Note that this means Java isn’t a purely object-oriented language.

**Q.19. Between Scala and Java, what are some features only one of them supports?**

* Operator overloading- Scala
* Static members- Java
* Primitive data types- Java
* Implicits and traits- Scala
* Explicit type casting- Java
* break and continue statements- Java
* Checked and unchecked exceptions- Java
* Pattern matching- Scala
* ++ and — operators- Java

[**Let’s revise Features of Java**](https://data-flair.training/blogs/features-of-java/)

**Q.20. Can companion objects access private members of their companion classes?**

Yes. A companion object can access its companion class’ private members, but the vice-versa is true as well.

**Scala Interview Questions and Answers for freshers- Q. 11,12,13,16,18,19,20**

**Scala Interview Questions and Answers for experienced- Q. 14,15,17**

**Q.21. How do you implement interfaces in Scala?**

This is a trick question to throw you off. Scala has no interfaces. Here, we have traits instead. Refer to [**Traits**](https://data-flair.training/blogs/scala-trait/) and [**Trait Mixins in Scala**](https://data-flair.training/blogs/scala-trait-mixins/).

**Q.22. Functions and methods are the same in Scala. Is that true?**

No. While we use the same syntax to define both, there is a difference:

* Methods associated with an object. So we can call it on an instance of a class, but cannot call it directly without an object. We can define it in a class or in a trait.
* A function, however, doesn’t associate with a class or with a trait. We can access it without an object, and we define it in a Scala package.

[**Let’s learn Scala String Method**](https://data-flair.training/blogs/scala-string-method/)

**Q.23. What are some popular MVC frameworks for developing web applications with Scala?**

Here are a few names:

* Play
* Spray
* Lift
* Scalatra

**Q.24. How many public class files can you define in a Scala source file? Not more than one?**

While that would be the case with Java, Scala actually lets us have as many as we want.

**Q.25. What are some default imports in Scala?**

Some names:

* java.lang package
* scala package
* scala.PreDef

In Java, the JVM automatically imports java.lang as the default package into each program so we don’t have to do it manually.

**Q.26. Between Java and Scala, what are some OOP concepts that only one of the languages supports?**

Scala has the following OOP constructs that Java lacks:

* Traits
* It solves the inheritance diamond problem

And Java has these:

* Interfaces
* Enum

**Q.27. How many operators does Scala have?**

Only one- The ‘=’ operator. No, we’re not forgetting +, -, and other such ones. Those are methods, not **[Scala operators](https://data-flair.training/blogs/scala-operator/)**. You can attribute this to operator overloading.

Let’s take an example. For the expression 5+6, + is a method in the Int class. The compiler knows that these are integers, and looks for the + method in Int. It makes a call on the object 5 as: 5.+(6), and returns 11. This is some syntactic sugar.

**Q.28. Name some IDEs that support Play and Scala-based Application Development.**

Two popular [**IDEs**](https://data-flair.training/blogs/scala-environment-setup/) that help us with our purpose are:

* IntelliJ IDEA
* Eclipse IDE

These IDEs have plugins for Scala. To find out how to download and install IntelliJ, read up on Scala Environment Setup.

**Q.29. Have you heard of SBT?**

Yes. SBT expands to Scala Build Tool. Using this simple build tool, we can easily develop Scala-based applications. We can also use it to develop for Play and Scala applications. IntelliJ uses SBT as a build tool for its Scala plugin.

**Q.30. Okay, so what are some other tools available for this purpose?**

While SBT is very popular, other common tools are Maven and Gradle.

**Scala Interview Questions and Answers for freshers- Q. 21,22,23,24,25,27,29,30**

Top Scala Interview Questions and Answers

Below, we are discussing some frequently asked Scala Interview Questions and Answers:

**Q.1. What is type inference in Scala?**

The Scala compiler decides data types or result types of elements like variables, expressions, and objects at compile-time.   
Scala infers the type for an expression fully or partially. This makes it work even when we don’t declare it explicitly. It also allows for type checking without type annotations. The compiler considers types of atomic values or subexpressions and aggregates it to decide a type for the whole expression.

[**For more on this, read up on Features of Scala.**](https://goo.gl/lZJOWF)

**Q.2. Is Int in Scala the same as java.lang.Integer in Java?**

While the two are similar in some ways, they also differ in other ways:

**Differences:**

* Int does not implement the Comparable interface; java.lang.Integer does.

**Similarities:**

* They’re both classes
* They are 32-bit signed integers
* They represent integer numbers

**Q.3. What is an anonymous function?**

An anonymous [**function in Scala**](https://data-flair.training/blogs/partial-functions-scala-guide/) is a function literal. At runtime, Scala instantiates into a function value. Since this is a way to create a function using just one line of code, we can call it lightweight. Here’s an example:

1. scala> val mul=(a:Int,b:Int)=>a\*b
2. mul: (Int, Int) => Int = $$Lambda$1541/400146874@3aca06a3
3. scala> **mul**(3,4)

res50: Int = 12

**Q.4. Explain implicit parameters.**

Much like default parameters, implicit parameters help us find default values. To make a parameter implicit, we can pass it to a method or a constructor and label it as *implicit*. So when we don’t mention the value of a parameter, the compiler searches for an implicit value defined within a scope.

In Part I, we discussed the precedence for implicit parameters.

**Q.5. What is a case class?**

[**Scala Case classes**](https://data-flair.training/blogs/scala-case-class/) let us model immutable data. For object construction, these classes have an apply() method to handle object construction. These classes have all *val*s, and so prove useful with pattern-matching. Let’s take an example.

1. scala> case class **Book**(title:String, author:String, ISBN: String)
2. defined class Book

**Q.6. What can you tell me about the Scala REPL?**

While we often call it ripple, REPL expands to Read Evaluate Print Loop. We can access it from the command prompt by typing ‘scala’:

1. C:\Users\lifei>scala

Welcome to Scala 2.12.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0\_161).

Type in expressions for evaluation. Or try :help.  
scala>

This is the Command Line Interface(CLI) and command-line shell for Scala, and we can execute Scala code in it as if in an interpreter. We can build and test small pieces of code in it.

**Q.7. What is a Map in Scala?**

A map is a collection in Scala that holds key-value pairs. A map can only hold unique keys, but values can be repeated. We also call these hash tables.

By default, maps are immutable, but we can use the scala.collection.mutable.Map class for a mutable map. This is an example:

1. scala> var m=**Map**("red"->0,"green"->1,"blue"->2)
2. m: scala.collection.immutable.Map[String,Int] = **Map**(red -> 0, green -> 1, blue -> 2)

[**For more on maps, read up on Maps in Scala.**](https://data-flair.training/blogs/scala-map/)

**Q.8. Differentiate between arrays and lists in Scala.**

1. scala> var m=**Map**("red"->0,"green"->1,"blue"->2)

We have the following differences:

* [**Arrays**](https://data-flair.training/blogs/scala-array/) are mutable; lists are immutable
* Arrays are invariants; lists are covariants
* In size, arrays are fixed, but lists are variable
* We can modify array values later, but that is not the same with lists

**Q.9. Is Scala a pure object-oriented language?**

Let’s see the [**object-oriented language.**](https://data-flair.training/blogs/scala-object-oriented-programming/)

* There are no static members
* Scala treats functions and primitives as objects

So, we can conclude that Scala is indeed purely object-oriented as a language.

**Q.10. What is a lazy val?**

Scala executes a val when we define it, but executes a lazy val only when we access it the first time. We declare it using the keyword *lazy*:

1. scala> lazy val x=7
2. x: Int = <lazy>
3. scala> lazy val x=2+3
4. x: Int = <lazy>

Where eager evaluation is at compile-time, lazy evaluation is at run-time.\

**Scala Interview Questions and Answers for freshers- Q. 1,3,4,5,7,8,9,10**

**Scala Interview Questions and Answers for experienced- Q. 2,6**

**Q.11. How many reserved keywords of Scala can you recall?**

Scala has 39 keywords-

abstract, case, catch, class, def, do, else, extends, false, final, finally, for, forSome, if, implicit, import, lazy, match, new, null, object, override, package, private, protected, return, sealed, super, this, throw, trait, try, true, type, val, var, while, with and, yield.

[**Let’s learn Scala Throw Keyword**](https://data-flair.training/blogs/scala-throw-keyword/)

**Q.12. Is it possible to use a Scala keyword as an identifier?**

While it might instinctively seem otherwise, you can use a reserved keyword as an identifier in Scala. Use backticks for this. As an example, we can use backticks on the word *yield* to use Java’s *yield* method from Thread class instead of Scala’s yield.  
Thread.`yield`()

**Q.13. Is an expression the same as a statement in a Scala?**

A statement is one or more operation, but an expression is a value. This means we can assign an expression to a variable, but we can’t assign a statement to the same. Examples of expressions- Scala’s if-condition, Java’s ternary operator. Examples of statements- Java’s if-condition.

[**Let’s revise Scala Regular expression**](https://data-flair.training/blogs/scala-regex/)

**Q.14. Differentiate between equals() and == in Scala. Is the latter the same as == in Java?**

When we try to compare two instances with the == operator, Scala calls the object’s equals() method. We use it to check instance equality. However, in Java, we use it to check reference equality. This tells us whether two references point to the same object.

**Q.15. Explain the differences between the if-else statements for Scala and Java.**

There is one main difference. This statement is an expression in Scala, but not in Java. There, we cannot assign it to a variable because it doesn’t return a value.

1. Scala:
2. scala> val x=3
3. x: Int = 3
4. scala> val kind= **if**(x%2==0) "even" else "odd"
5. kind: String = odd
6. Java:
7. int x=3;
8. String kind;
9. **if**(x%2==0)
10. kind=”even”;
11. else
12. kind=”odd”;

[**Learn more about Scala If-else Statements**](https://data-flair.training/blogs/scala-else-statements-statements/)

**Q.16. How are Scala’s inner classes different from those of Java?**

In Scala, an inner class is associated with the outer class’ object. In Java, however, an inner class is associated with the outer class; the inner class is a member of the outer class.

**Q.17. Is Scala expression-based or statement-based? What about Java?**

Yes, Scala is expression-based. You already know everything is a value in Scala. This means all expressions and statements evaluate to a value. So, we can assign expressions, functions, closures, objects, and other entities to variables. But things aren’t the same with Java. Hence, Java is statement-based.

**Q.18. Why doesn’t Scala have the keyword ‘static’?**

The team made this decision to maintain the nature of Scala as a pure object-oriented language. If they included it, we could access class members without having to create an object. That would violate the principles of OOP. Note that this means Java isn’t a purely object-oriented language.

**Q.19. Between Scala and Java, what are some features only one of them supports?**

* Operator overloading- Scala
* Static members- Java
* Primitive data types- Java
* Implicits and traits- Scala
* Explicit type casting- Java
* break and continue statements- Java
* Checked and unchecked exceptions- Java
* Pattern matching- Scala
* ++ and — operators- Java

[**Let’s revise Features of Java**](https://data-flair.training/blogs/features-of-java/)

**Q.20. Can companion objects access private members of their companion classes?**

Yes. A companion object can access its companion class’ private members, but the vice-versa is true as well.

**Scala Interview Questions and Answers for freshers- Q. 11,12,13,16,18,19,20**

**Scala Interview Questions and Answers for experienced- Q. 14,15,17**

**Q.21. How do you implement interfaces in Scala?**

This is a trick question to throw you off. Scala has no interfaces. Here, we have traits instead. Refer to [**Traits**](https://data-flair.training/blogs/scala-trait/) and [**Trait Mixins in Scala**](https://data-flair.training/blogs/scala-trait-mixins/).

**Q.22. Functions and methods are the same in Scala. Is that true?**

No. While we use the same syntax to define both, there is a difference:

* Methods associated with an object. So we can call it on an instance of a class, but cannot call it directly without an object. We can define it in a class or in a trait.
* A function, however, doesn’t associate with a class or with a trait. We can access it without an object, and we define it in a Scala package.

[**Let’s learn Scala String Method**](https://data-flair.training/blogs/scala-string-method/)

**Q.23. What are some popular MVC frameworks for developing web applications with Scala?**

Here are a few names:

* Play
* Spray
* Lift
* Scalatra

**Q.24. How many public class files can you define in a Scala source file? Not more than one?**

While that would be the case with Java, Scala actually lets us have as many as we want.

**Q.25. What are some default imports in Scala?**

Some names:

* java.lang package
* scala package
* scala.PreDef

In Java, the JVM automatically imports java.lang as the default package into each program so we don’t have to do it manually.

**Q.26. Between Java and Scala, what are some OOP concepts that only one of the languages supports?**

Scala has the following OOP constructs that Java lacks:

* Traits
* It solves the inheritance diamond problem

And Java has these:

* Interfaces
* Enum

**Q.27. How many operators does Scala have?**

Only one- The ‘=’ operator. No, we’re not forgetting +, -, and other such ones. Those are methods, not **[Scala operators](https://data-flair.training/blogs/scala-operator/)**. You can attribute this to operator overloading.

Let’s take an example. For the expression 5+6, + is a method in the Int class. The compiler knows that these are integers, and looks for the + method in Int. It makes a call on the object 5 as: 5.+(6), and returns 11. This is some syntactic sugar.

**Q.28. Name some IDEs that support Play and Scala-based Application Development.**

Two popular [**IDEs**](https://data-flair.training/blogs/scala-environment-setup/) that help us with our purpose are:

* IntelliJ IDEA
* Eclipse IDE

These IDEs have plugins for Scala. To find out how to download and install IntelliJ, read up on Scala Environment Setup.

**Q.29. Have you heard of SBT?**

Yes. SBT expands to Scala Build Tool. Using this simple build tool, we can easily develop Scala-based applications. We can also use it to develop for Play and Scala applications. IntelliJ uses SBT as a build tool for its Scala plugin.

**Q.30. Okay, so what are some other tools available for this purpose?**

While SBT is very popular, other common tools are Maven and Gradle.

**Scala Interview Questions and Answers for freshers- Q. 21,22,23,24,25,27,29,30**

 Scala Interview Questions With Answers

Below, we are discussing some best Scala Interview Questions with Answers:

**Q.1. How is a trait different from an abstract class?**

* A class can inherit from only one [**abstract class**](https://data-flair.training/blogs/scala-abstract-class/), but from multiple traits
* Abstract classes are fully interoperable with [**Java**](https://data-flair.training/blogs/java-tutorial/); traits- only when they hold no implementation code
* Abstract classes can have both- constructor parameters and type parameters; traits only have type parameters

**Q.2. How does Scala support both- highly scalable and high-performance applications?**

Not only does Scala use the Actor Concurrency Model, it also supports multi-paradigm programming(object-oriented and functional). This lets us develop highly scalable and high-performance applications.

**Q.3. Differentiate between a Java future and a Scala future.**

There’s one main difference between java.util.concurrent.Future and scala.concurrent.Future. Working with Java’s Future, you must access results using a blocking get() method. With Scala’s future, you can attach callbacks to complete. You can also map it and monadically chain Futures together without blocking. For a Java Future, you can call the method isDone() to confirm whether a Future has completed. This lets you avoid blocking.

**Q.4. Explain the differences between Option, Try, and Either.**

Option, Try, and Either are monads. We can use them to represent computations gone awry. But how are they different? Here we go:

* [**Options**](https://data-flair.training/blogs/scala-option/) denote absence of value. We can use them when we want to perform a search. Example- database access
* Try wraps runtime exceptions. Hence, we can consider it as a monadic approach to the try-catch block in Java.
* Either comes in handy when we must provide more information about the failure of the computation. Eithers have two possible return types- successful/correct/expected and the error case.

**Q.5. List some advantages of functional programming.**

Functional programming has the following advantages:

* Modular
* Easier to test
* Easier to understand
* Supports reuse
* Less prone to bugs
* Supports parallelism and generalization

[**Read about Scala Functions**](https://data-flair.training/blogs/scala-functions/)

**Q.6. What is the equivalent type for java.lang.Object here in Scala?**

Where there’s java.lang.Object in Java, we have AnyRef in Scala. This is in context of a Java Runtime Environment (JRE).

[**Have a look at JRE vs JVM**](https://data-flair.training/blogs/jdk-vs-jre-vs-jvm/)

**Q.7. Mention the equivalent construct for Scala’s Option in Java SE 8.**

For[**Option in Scala**](https://data-flair.training/blogs/scala-option/), we have Optional in Java. This is a class that lets us represent existing or non-existing values. We can access it using the java.util package.

We can use these constructs to represent optional values. We can also avoid unwanted null checks and NullPointerException.

**Q.8. Define covariance and contravariance.**

A generic type is covariant if it has the same subtype relationship as its type parameter T. It is contravariant when its subtype relationship is in the reverse of that of its type parameter.

**Q.9. Explain the Either/Left/Right design pattern in Scala.**

Either is an abstract class. With it, we can represent a value of two possible kinds. The two parameters it takes are Either[A, B]. The two subtypes it has are Left and Right. Left is an instance A and Right is an instance B.

**Q.10. Now, explain the Option/Some/None design pattern.**

Option, in Scala, is an abstract class with two subclasses Some and None. Bounded collections like these can hold either one element or none. When it holds one element, it is a Some; otherwise, it is a None. While Some represents an existing value, None represents absence of a value. Here, None is an object and Some is a case class. This also lets us use them in pattern matching.

Together, these definitions form an Option/Some/None design pattern in Scala.

**Scala Interview Questions With Answers for Freshers – Q. 1,2,3,4,5,8,9**

**Scala Interview Questions with Answers for Experienced – Q. 6,7,10**

**Q.11. List down predefined value types in Scala.**

Scala has 9 predefined, non-nullable value types under AnyVal. These are-

Double, Float, Long, Int, Short, Byte, Unit, Boolean, and Char.

[**Follow the link to learn more about Scala Data Types**](https://data-flair.training/blogs/scala-data-types/)

**Q.12. Which design pattern does pattern matching follow in Scala?**

It follows the Visitor design pattern. It is a behavioral design pattern to help perform an operation on a bunch of similar objects. Another operator that follows this design pattern is Java’s *isinstanceof* operator.

[**Learn pattern Matching in Scala**](https://data-flair.training/blogs/scala-pattern-matching/)

**Q.13. What is a Guard in a for-comprehension?**

When, in a for-comprehension, we want to filter elements, we can use an if-condition. We can call this if-condition a ‘guard’. When a guard is True, Scala adds that element to a new collection. Otherwise, it performs no operation. Let’s take an example.

1. scala> val nums=**List**(0,1,2,3,4,5,6,7,8,9,10)
2. nums: List[Int] = **List**(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
3. scala> **for**(num<-nums if num%3==0) yield num

**res0:** List[Int] = List(0, 3, 6, 9)

This code declares a List and adds to another List only those elements which are divisible by 3.

**Q.14. Explain type-casting in Scala.**

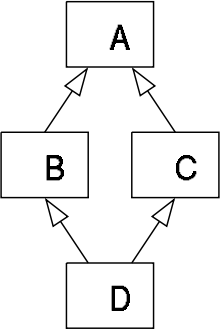
We can cast values in an order:

Byte-> Short-> Int-> Long-> Float-> Double

This order is unidirectional; anything else makes the compiler throw an error. It is possible to cast a Char into an Int. It is also possible to cast a reference type to a subtype.\

**Q.15. Explain the Diamond Problem.**

This takes us to multiple inheritance. The deadly diamond is when one class extends more than one trait holding the same method.

[](https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2018/07/diamond.png)

*Scala Interview Questions with answers – Dimaond Problem*

Scala resolves this by following rules called *Class Linearization*. Let’s take an example.

1. scala> trait A{
2. | def **show**(){
3. | **println**("A's show")
4. | }
5. | }
6. defined trait A
7. scala> trait B extends A{
8. | override def **show**(){
9. | **println**("B's show")
10. | }
11. | }
12. defined trait B
13. scala> trait C extends A{
14. | override def **show**(){
15. | **println**("C's show")
16. | }
17. | }
18. defined trait C
19. scala> class D extends B with C{}
20. defined class D
21. scala> val d=new **D**()
22. d: D = D@72d4d9ac
23. scala> d.**show**()

C’s show

As you can see, the compiler prints C’s message. The compiler reads “class D extends B with C” and reads it right to left. It takes the show() method from the left-most trait, which, to it, is C.

**Q.16. What is a Range?**

Range is a kind of a collection in Scala. It is a lazy collection and we can use it to denote an ordered sequence of Integer values. To use it, we can use the class scala.Range. Here’s an example.

1. scala> 1 to 7

**res4:** scala.collection.immutable.Range.Inclusive = Range 1 to 7

1. scala> 1 until 7

**res5:** scala.collection.immutable.Range = Range 1 until 7

1. scala> **for**(i<-1 to 7) **println**(i)

1  
2  
3  
4  
5  
6  
7

**Q.17. Is it possible to set default values for class parameters?**

When a caller omits a few parameters, Scala will take in the default values if we have provided them. It then becomes optional to include these parameters in a call. It is also possible to provide default values to method parameters.

1. scala> class **A**(name:String="Ayushi"){}
2. defined class A

**Q.18. What do you think is a pure function?**

A pure function has no side-effects. But what does that mean? Well, even if you call it a thousand times, it will return the same outputs for respective inputs.  
\* is a pure function with Scala. Let’s see if this always gives us the same result.

1. scala> 3\*4

**res8:** Int = 12

1. scala> 3\*4

**res9:** Int = 12

1. scala> 3\*4

**res10:** Int = 12

1. scala> 1\*2

**res11:** Int = 2

1. scala> 3\*4

**res12:** Int = 12

It does. Hence, we conclude what a pure function is.

[**Learn about Scala Currying Functions**](https://data-flair.training/blogs/scala-currying/)

**Q.19. What are some rules for Scala named arguments?**

There are two rules-

* You can reorder named arguments
* For a parameter list with both named and unnamed arguments, the latter must appear first. These should be in order of their parameters in the method signature.

1. scala> def **show**(name:String,surname:String):Unit={
2. | **println**(name+" "+surname)
3. | }
4. show: (name: String, surname: String)Unit
5. scala> **show**("Ayushi",surname="Sharma")

Ayushi Sharma

**Q.20. How is a function different from a procedure?**

A function is a computation unit and has no side effects. A procedure is a computation unit too, but has side effects.  
However, both help with computation; this is how they’re similar.

**Scala Interview Questions With Answers for Freshers – Q. 11,12,13,16,18,19,20**

**Scala Interview Questions with Answers for Experienced – Q. 14,15,17**

**Q.21. It is impossible to mark constructor parameters as private in Scala?**

No, it isn’t. We can declare a parameter to be private if we declare it without a *var* or a *val* keyword. Then, we can only access these values within the class.

[**Read more about Scala constructor**](https://data-flair.training/blogs/scala-constructor/)

**Q.22. We have seen that traits let us implement multiple inheritance. Can traits extend other traits, though?**

Yes, they can. We can extend a trait using the *extends* keyword. And if we make a class extend it, we can use the keyword *override* to implement its abstract members.

**Q.23. Mention a way to import all classes from a package.**

While we can import a single class from a package using the dot operator and the name of that class. But to avoid having to name each class that we want to import, we can rather import all classes at once. We use the underscore for this.  
import sound.\_

This imports all classes. But we can also import some certain classes at once.

import sound.(Treble, Soprano, Bass}

**Q.24. Can classes in Scala extend more than one class?**

A class can extend only one class. So, no, a class cannot extend more than one class. However, it can have many mixins.

**Q.25. Can you Subtype using a trait?**

Absolutely. We can use a subtype of a trait where we need that trait.

**Q.26. So, what is a mixin?**

A **[mixin](https://data-flair.training/blogs/scala-trait-mixins/)**, in Scala, is a trait that we can use to compose a class. It is also true that a class can have more than one mixin. To add a mixin to a class, we use the *with*keyword as we have seen earlier in our tutorials.

**Q.27. How is nesting functions helpful in Scala?**

When we nest functions, it lets us structure code. This also promotes readability.

**Q.28. What is an abstract type?**

We can use the *type* keyword to define an abstract type in Scala. This describes element type, but only its implementation defines its actual type.

**Q.29. Explain sealed classes in Scala.**

When we mark a trait or a class as *sealed*, we must declare all subtypes in the same file. This way, we can make sure that we know all subtypes.

1. scala> sealed abstract class Transport
2. defined class Transport
3. scala> case class **Scooter**() extends Transport
4. defined class Scooter
5. scala> case class **Car**() extends Transport
6. defined class Car
7. scala> def **show**(vehicle:Transport):String=vehicle match{
8. | case a: Scooter=>"Two wheels"
9. | case b: Car=>"Four wheels"
10. | }

**show:** (vehicle: Transport)String

**Q.30. Can you reassign parameters for case classes?**

No, we cannot, since case classes help model immutable data. Its parameters are implicitly *public val*. We could use *var*s here, but Scala discourages doing so.

**Scala Interview Questions With Answers for Freshers – Q. 22,24,26,27,28,29**