Scala: A History

In this lesson, you will be introduced to the Scala programming language and its multiple paradigms.

We'll cover the following

- Roots
- A Multi-Paradigm Programming Language
- Scala: Through the Eyes of Java

Roots

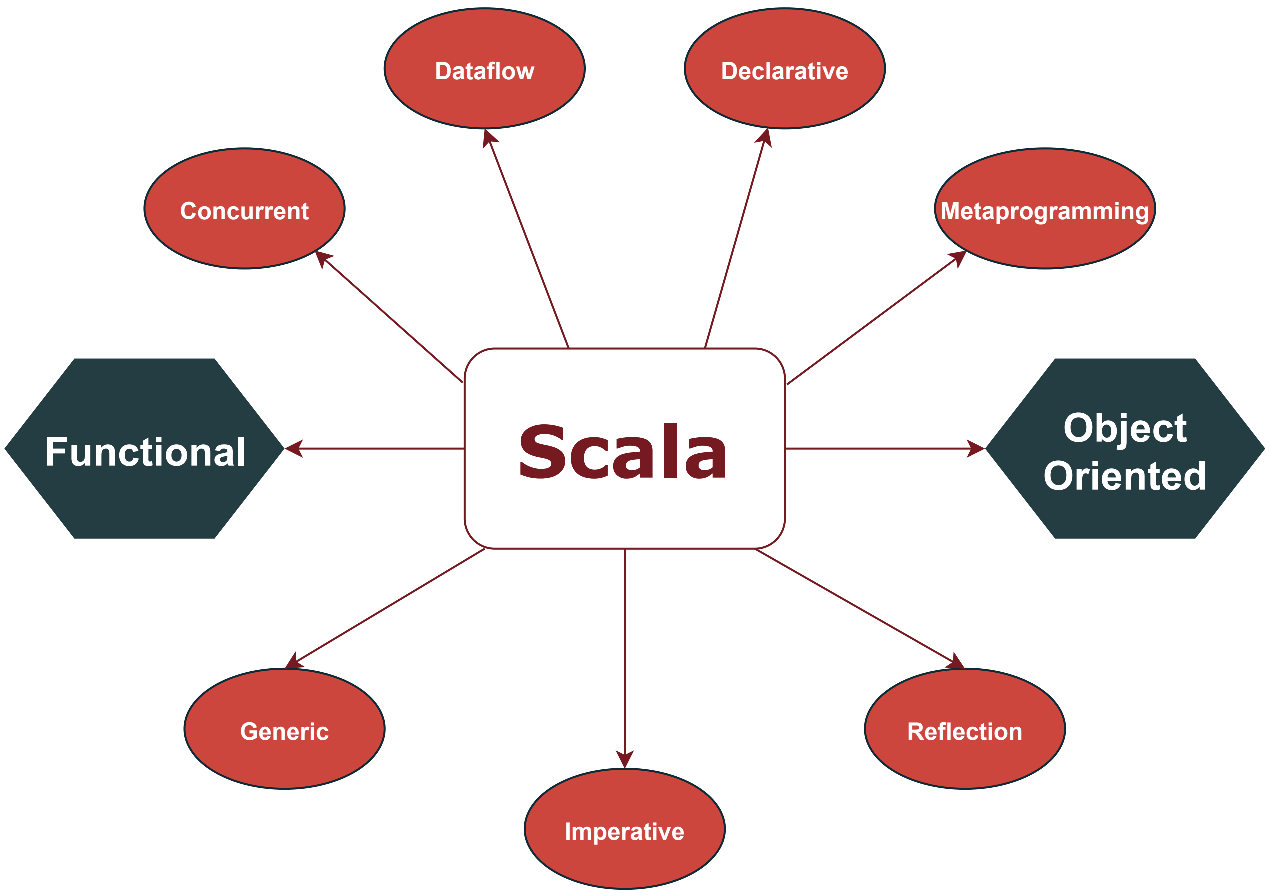
Scala is a modern programming language designed and created by Martin Odersky. The design of the language started in 2001 and was released to the public in early 2004. Martin Odersky had a huge hand in the implementation of javac (the primary Java compiler) and also designed Generic Java, a facility of generic programming that was added to the Java programming language in 2004. This is why it doesn't come as a surprise that Scala is similar to Java in many aspects, it's actually written to run in JVM (Java Virtual Machine).

Scala's design was influenced by multiple programming languages and ideas in programming research. In fact, Martin Odersky himself has stated that very few features of Scala can actually be labeled 'new', and in Scala, "innovation comes primarily from how its constructs are put together." It was basically designed to be a "better language."

A Multi-Paradigm Programming Language

Scala is a multi-paradigm language, which is just as much object-oriented as it is functional. But what does it mean for a programming language to be *multi-paradigm*?

Every language follows a set of rules or a pattern known as a paradigm and can be classified according to which paradigm it falls in. A paradigm characterizes the style, concepts, and methods of the language for describing situations and processes and for solving problems. Hence, each paradigm has a particular application it is best suited for.



Scala's Multiple Paradigms

Some common paradigms include imperative programming, declarative programming, object-oriented programming, and functional programming, to name a few. While there are very few languages that purely support a single paradigm, most languages are primarily known for a particular paradigm regardless of supporting others. For instance, Haskell is known as a Functional Programming language but also supports imperative programming.

If a language is designed to allow programming in multiple paradigms and cannot be distinctively classified under one category, it is known as a multi-paradigm language. As Leda designer Timothy Budd puts it: "The idea of a multi-paradigm language is to provide a framework in which programmers can work in a variety of styles, freely intermixing constructs from different paradigms." This equips the programmer with a larger number of tools to solve a wider range of problems.

Scala: Through the Eyes of Java

Martin Odersky's primary influence when developing Scala was the Java programming language, so much so that Scala is actually written to run in Java Virtual Machine. Scala programs compile to JVM bytecodes, and their runtime performance is mostly on par with Java programs. If you are not familiar with bytecode, that's perfectly fine. All you need to know is that Java programs don't compile into executable files; instead, they compile into bytecodes which are then executed during runtime.

Scala allows you to call Java methods, access Java fields, inherit from Java classes, and implement Java interfaces in your code without the added requirement of any special syntax. Programmers are not even aware of how heavily Scala code makes use of Java libraries.

Scala also reuses Java types. Scala's `Int` is represented as Java's `int`, Scala's `Float` is represented as Java's `float`, and so on. Scala's string literals are of type `java.lang.String` and Java's string methods can be called on them.

While the parallels between Java and Scala are evident, Scala differs in many ways as well with its code being much more concise, clean, and with its own benefits.

In the next lesson, we will learn about Scala's scalable nature by exploring its two main paradigms.