References:

[Introduction | Tour of Scala | Scala Documentation (scala-lang.org)](https://docs.scala-lang.org/tour/tour-of-scala.html)

[The Scala Programming Language (scala-lang.org)](https://www.scala-lang.org/)

[Java Memory Management - GeeksforGeeks](https://www.geeksforgeeks.org/java-memory-management/)

[Baeldung on Scala](https://www.baeldung.com/scala/)

[Scala Grammar (-scala-lang.org)](https://www.scala-lang.org/files/archive/spec/2.11/13-syntax-summary.html)

[Scala for the Impatient: Horstmann, Cay: 8601300203072: Books (amazon.com)](https://www.amazon.com/Scala-Impatient-Cay-S-Horstmann/dp/0321774094/ref=as_li_ss_tl?ie=UTF8&qid=1440439813&sr=8-5&keywords=scala&pebp=1440439817501&perid=13YYSDWKJ6PS214DKKYY&linkCode=sl1&tag=nethta-20&linkId=8a1ce7100034606d5f44abf10fd15bdd)

[(389) Scala Tutorial - YouTube](https://www.youtube.com/watch?v=DzFt0YkZo8M&t=3259s)

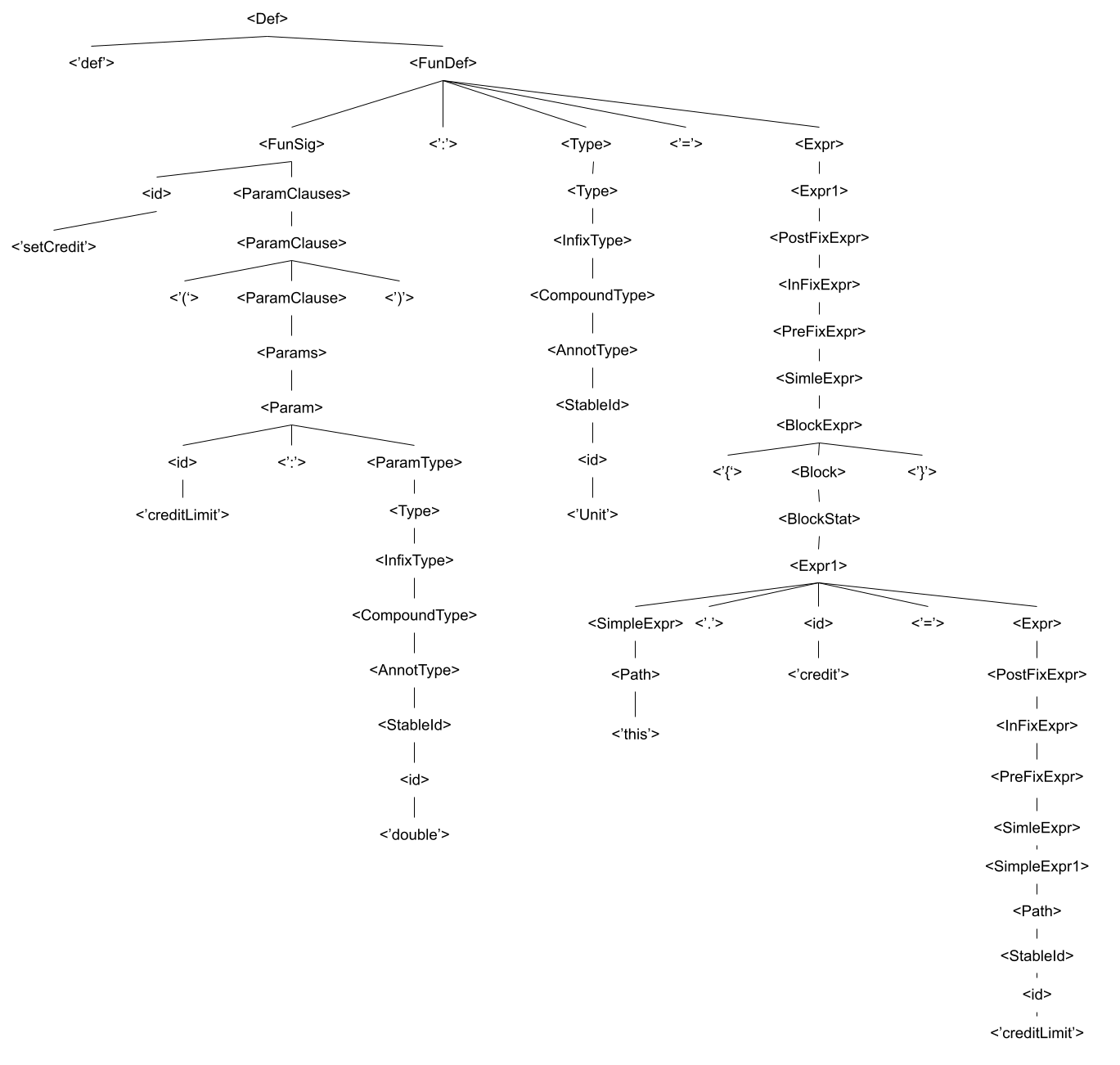
* The context and primary purpose of the language. When was it created and why? - Matt Tobeck

Development of Scala began in 2001 at the École Polytechnique Fédérale de Lausanne in Switzerland and was spearheaded by Martin Odersky. The initial goal with Scala was to show that “... a fusion of functional and object-oriented programming is possible and practical.” said Odersky in an interview with JAXenter in 2016. Having met this goal, Scala is actually a multi-paradigm language because it has elements of Object-Oriented, Functional, and Imperative programming languages.

Scala is very much object-oriented because of the fact that it was largely derived from the OO language Java. This means that Scala inherited many of the OO features of Java such as inheritance through the use of classes, polymorphism (parametric, subtype, overloading, but no coercion), and encapsulation. One of the biggest differences between Scala and other OO languages is as previously mentioned also a functional language which allows it to support standard functional features such as nested method definitions, pattern matching, higher order functions, as well as being optimized for tail recursion. Conversely, Scala is different from many other functional languages because of its OO properties mentioned above.

The following link is to the grammar for Scala provided by Scala-Lang: [Scala Grammar](https://www.scala-lang.org/files/archive/spec/2.11/13-syntax-summary.html)

And here are three parse trees for the grammar that illustrate various unique features of the Scala language:



• Subroutines, including any overloading or polymorphism – Awais Nadeem

Functions/subroutine:

Since Scala is a functional language as well, it supports a menagerie of features with relations to functions. For functional features, Scala supports the ability to create & define functions, anonymous functions, pass/store functions as parameters, return functions, curry parameters, nest functions, pattern matching, higher order functions, and of course recursion. A Scala function definition consists of an optional identifier (such override), optional scope modifier (public by default), the name of the method, the parameters (can specify zero or many) alongside their type (can be the type Any), and finally the return type (Unit in case of void). To define an anonymous function, specify the parameters in a tuple alongside their type and use the => to indicate the operation. Define the operation on 1 line, but incase you need more, you can do use curly braces as => { function definition here }. You can very easily assign functions to variables and treat them as variables – in fact everything is a type/object in Scala, so the assignment is identical. Since functions are just variables they can be treated as parameters into functions, values out of functions, etc. Like ML and other functional languages, Scala supports currying and has built in functions as foldr, foldl, map. Like ML, Scala can have function definitions inside functions and both can call each other, parent & child’s can call each other, and even do intertwined/interleaved recursion. Scala supports pattern matching in a form similar to switch statements. It supports wild card matching, tuple matching, both for functions & this can also be used as a regular switch statement (it is also used in try catch expressions). Finally, Scala does have the ability to pass functions into functions and form higher order functions. Scala can also be optimized for tail recursion (this is supported via a decorator @tailrec).

Scala supports all forms of polymorphism. Being an object-oriented language derived from Java, Scala of course supports Parametric Polymorphism (generics) and subtype polymorphism (execution time check instead of link time on subtype object function call). For example, our linked list program illustrates how Scala is able to create linked lists of any objects, while subtype polymorphism is illustrated in our bank program with the student object using a generic Account (base class) to do transactions with both credit and debit accounts. The doTransactions() method calls abstract methods getBalance(), deposit(), and withdraw(), which are override in subclasses appropriately. Scala also supports overloading – i.e., functions with the same signature but different formal parameters. Finally, Scala does support parameter coercion, for example in our trivial program we can do sum (which accepts double) with integers and the programs still works as Scala automatically type casts implicitly.

• Parameter passing – Awais Nadeem

Scala’s parameter passing is by value exclusively. In call-by-value, the passed in arguments are evaluated once before execution of the function. However, everything is a value in Scala, so this does not tell the entire story. For objects and data structures such as Arrays, Scala does not pass the entire array or object as a value to functions. Instead, it passes the reference as a value to the function. This is identical to Java - when a value of a reference is passed, changes to the object are immediate and modify the actual object. Scala is very strongly typed and static. Like ML, Scala tries to ensure that there are no side effects. For example, our linked list program contains an insert method which gets passed along an index. In a language like Java, you can just modify the index variable and do something similar to:

While(index >0){

Index –;

Temp.next;

}

However, in Scala, this is a reassignment of the variable index. To accomplish this goal, one must copy the variable in a local var parameter and traverse list accordingly. Finally, since Scala is a functional language and supports higher order functions, Scala is capable of by-name parameter passing. This is the most flexible type of parameter passing and is basically identical to passing an anonymous function that returns the value of the variable in the caller’s context every time it is used.

• Memory management – Awais Nadeem

Since Scala is derived from Java and runs in the JVM, its memory management is identical to Java. Java’s memory is divided in the Heap area, Method Area, JVM Stack, Native Method Stack, and PC Registers. For the Heap area, this is whenever we create objects in Java (i.e., whenever we use the new keyword). Since everything is an object in Scala, we make extensive use of this. Next is the method area, this is where the methods of objects are stored. This area is most likely used for Scala unique with its traits as traits can possess implemented methods and this area is responsible for containing constructors, class structures, field data, and interfaces in Java. The JVM stack is the call stack for Java programs, this is indifferent for Scala. The native call stack is the call stack for native functions (OS system calls, interrupts, io, etc.) – this again is OS specific and unrelated to Scala uniqueness in derivation. Finally, the program counter just keeps track of the currently executing function. Scala does support garbage collection identical to Java. Java uses mark and sweep among other algorithms (concurrent mark & sweep, serial/parallel garbage collection, G1) for its garbage collection and so Scala would be the same with this regard. Scala will also suffer from the same issues of memory management as java (high memory use) and will give the same errors in relation to memory management as Java.

• Records/object – Awais Nadeem

Scala is a pure object language meaning everything is encapsulated into an object. Despite being derived from Java, Scala does not have primitives, however, it is nearly identical in all other regards. Like Java, Scala supports the creation of classes & objects. You can create a class identical to Java, however, you need not write a constructor. Instead, the constructor is the signature/declaration of the class – you specify all instance variable, types, defaults inside of it. This shows how Scala branches off Java in a clever way by keeping the already phenomenal object oriented of Java, while also trying to improve upon the annoyances of Java. Scala supports all the OOP features of Java but adds its own custom functionality. For example, in Java you can extend multiple interfaces – but interfaces must not have implementation. In Scala, however, you can extend multiple traits which are identical to interfaces but can have implementations & instance variables. Scala also has extended identifier such as object-private to help programmers finetune their information hiding. Scala has objects definitions which are singleton objects - these are used very commonly so it’s nice for Scala to provide a readily available implementation. These objects can be useful to provide extended encapsulation on already existing classes, while also being used as a potential namespace Scala also tries to aid programmers with OOP. For example, Scala allows the specification of case classes in which Scala tries to take care of some implementation details for you. By using a case class, Scala will create a apply method which will handle the creation of objects for you (this means you do need to use new keyword to create objects). It will also provide common implementations such as .equals() method which will compare structure equivalence (most often the desired implementation).

Instruction to install and run Scala:

Online Scala runner: scastie.scala-lang.org

Prerequisite: JVM installed & operational

Download and installer Coursier: <https://github.com/coursier/launchers/raw/master/cs-x86_64-pc-win32.zip>

Extract and run exe (Scala should be installed, you might need to restart explorer/terminal & or verify environmental path)

Go into cmd, index into where scala file resides.

To compile, do scalac <program>.scala

To execute, do scala <program>.scala