Scala Day 1

Scala Programming.

By Dhandapani Yedappalli Krishnamurthi Oct 6, 2025

Day 1: Scala Foundations for Data Engineers as a complete, *hands-on* training module.
Below is a full guide that includes **concept explanations, code samples, visual illustrations (described), and top GitHub references** to help you practice real-world data engineering with Scala + Spark.

🧠 Day 1 – Scala Foundations for Data Engineers

🎯 Learning Objectives

By the end of this session, you will:

- Understand how Scala fits into the Big Data ecosystem.
- Set up your environment for Spark development.
- Learn Scala syntax essentials (variables, control flow, functions).
- Practice functional transformations and pattern matching.
- Apply Scala to clean and transform datasets (CSV → JSON).

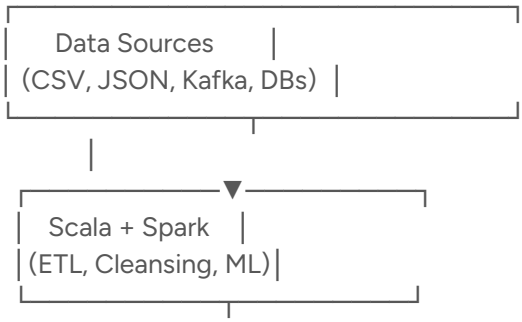
1️⃣ Overview: Scala in the Big Data Ecosystem

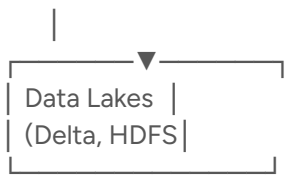
🔍 Why Scala?

Scala combines **functional + object-oriented** paradigms, making it ideal for **distributed data processing** with tools like:

Framework	Scala Role
Apache Spark	Native API language
Apache Kafka	Strong type-safety for stream processing
Apache Hadoop	Integration through MapReduce & HDFS clients
Hive / Delta Lake	ETL & transformation scripting
Akka / Flink	Reactive stream processing

👉 **Scala = "Java efficiency + Python flexibility + Functional programming"**
Illustration:





2️⃣ Environment Setup

⚙️ Install Dependencies

```
# Install JDK 11
sudo apt install openjdk-11-jdk -y
# Install Scala 2.12
sudo apt install scala -y
# Install SBT (Scala Build Tool)
sudo apt install sbt -y
# Download Spark 3.3.4 prebuilt with Hadoop 3
wget https://downloads.apache.org/spark/spark-3.3.4/spark-3.3.4-bin-hadoop3.tgz
tar -xvzf spark-3.3.4-bin-hadoop3.tgz
export SPARK_HOME=~/spark-3.3.4-bin-hadoop3
export PATH=$PATH:$SPARK_HOME/bin
# Verify
scala -version
spark-shell --version
```

IDE: Install IntelliJ IDEA (Community Edition) → Add Scala plugin.

SBT Project Structure

```
scala-foundations/
├── build.sbt
├── project/
├── src/
│   ├── main/
│   │   └── scala/
│   │       └── com/example/
│   │           └── Day1.scala
│   └── test/
│       └── scala/
└── data/
    ├── input.csv
```

Sample build.sbt:

```
name := "ScalaDataEngineeringDay1"
version := "0.1"
scalaVersion := "2.12.18"
libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-core" % "3.3.4",
  "org.apache.spark" %% "spark-sql" % "3.3.4"
)
```

3️⃣ Basic Syntax and REPL

Launch REPL:

Examples:

```
val name: String = "Dani"
var count = 10
println(s"Hello, $name! Count = $count")
// Immutable vs mutable
```

```
val x = 5   // immutable
var y = 10  // mutable
y += 5
println(y)
```

4️⃣ Variables, Data Types, and Operators

```
val a: Int = 10
val b: Double = 5.5
val result = a * b
println(result)
val msg = if (a > 5) "Big" else "Small"
println(msg)
```

Operators

```
println(a + b)
println(a == b)
println(a != b)
println(a > 5 && b < 10)
```

5️⃣ Control Structures

```
for (i <- 1 to 5) println(i)
val fruits = List("apple", "banana", "mango")
for (f <- fruits if f.startsWith("b")) println(f)
var i = 0
while (i < 3) {
  println(s"Index: $i")
  i += 1
}
val grade = 85
val remark = grade match {
  case x if x >= 90 => "Excellent"
  case x if x >= 75 => "Good"
  case _ => "Needs Improvement"
}
println(remark)
```

6️⃣ Functions and Recursion

```
def add(a: Int, b: Int): Int = a + b
def factorial(n: Int): Int =
  if (n == 0) 1 else n * factorial(n - 1)
println(add(5, 3))
println(factorial(5))
```

7️⃣ Collections and Functional Transformations

✅ Lists, Sets, Maps

```
val nums = List(1, 2, 3, 4, 5)
println(nums.map(_ * 2))
println(nums.filter(_ % 2 == 0))
println(nums.reduce(_ + _))
val unique = Set(1, 2, 2, 3)
val capitals = Map("India" -> "Delhi", "France" -> "Paris")
println(capitals("India"))
```

8️⃣ Tuples and Case Classes

```
val person = ("Dani", 30, "Data Engineer")
println(person._1)
```

```
case class Employee(name: String, age: Int, dept: String)
val emp = Employee("Asha", 28, "ETL")
println(emp.name)
```

9️⃣ Pattern Matching

```
def describe(x: Any): String = x match {
  case 0 => "Zero"
  case i: Int => s"Integer $i"
  case s: String => s"String $s"
  case _ => "Unknown"
}
println(describe(10))
println(describe("Spark"))
```

🔟 Error Handling (Option, Try, Either)

```
def safeDivide(a: Int, b: Int): Option[Double] =
  if (b == 0) None else Some(a / b.toDouble)
println(safeDivide(4, 2))
println(safeDivide(4, 0))
import scala.util.{Try, Success, Failure}
val res = Try(10 / 0)
res match {
  case Success(v) => println(v)
  case Failure(e) => println(s"Error: ${e.getMessage}")
}
```

🧪 Hands-on: Transforming & Cleaning Data (CSV → JSON)

📁 Sample data/input.csv

```
id,name,city
1,Dani,Chennai
2,Asha,Mumbai
3,Ravi,Bangalore
```

💻 Spark Scala Code (src/main/scala/com/example/Day1.scala)

```
package com.example
import org.apache.spark.sql.{SparkSession, functions => F}
object Day1 {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder()
      .appName("Day1-CSVtoJSON")
      .master("local[*]")
      .getOrCreate()
    // Step 1: Read CSV
    val df = spark.read.option("header", "true").csv("data/input.csv")
    // Step 2: Clean Data
    val cleaned = df.withColumn("name", F.initcap(F.col("name")))
                    .withColumn("city", F.upper(F.col("city")))
    // Step 3: Write as JSON
    cleaned.write.mode("overwrite").json("data/output_json")
    println("✅ Data transformation complete: CSV → JSON")
    spark.stop()
  }
}
```

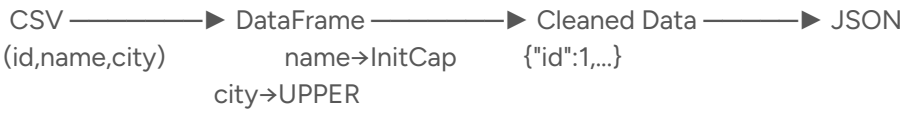
Run using SBT:

sbt run

Expected Output (JSON):

```
{"id":"1","name":"Dani","city":"CHENNAI"}
{"id":"2","name":"Asha","city":"MUMBAI"}
{"id":"3","name":"Ravi","city":"BANGALORE"}
```

🧩 Visualization (Concept Illustration)



🧭 Recommended Git Repositories

Purpose	Repository	Highlights
Scala for Spark Beginners	rockthejvm/scala-beginners	Clean examples of Scala syntax & functional programming
Spark ETL in Scala	databricks/spark-examples	Covers ETL, DataFrames, and structured streaming
Functional Scala	scala-exercises/exercises-scala tutorial	Hands-on exercises for Scala syntax, collections, and FP

💡 Homework / Practice Tasks

1. Modify the CSV→JSON code to:
   - Filter out records where city = "MUMBAI"
   - Add a new column country = "India"
2. Create a function to count how many records are from each city.
3. Handle empty values in CSV using na.fill.