🟦 1. Essential Scala Syntax

✅ Variables, Types

```
val name: String = "Dani"      // immutable
var age: Int = 30              // mutable

val pi: Double = 3.14
val isValid: Boolean = true
```

✅ Functions

```
def add(a: Int, b: Int): Int = a + b

println(add(10, 20))   // 30
```

✅ Loops, Conditionals

```
for (i <- 1 to 5) println(i)

val result = if (age > 18) "Adult" else "Minor"
println(result)
```

✅ Collections (List, Seq, Map)

```
val nums: List[Int] = List(1,2,3)
val seq: Seq[String] = Seq("a", "b", "c")
val mp: Map[Int, String] = Map(1 -> "one", 2 -> "two")

println(nums(0))     // 1
println(mp(2))       // two
```

✅ Tuples

```
val data = (1, "Arun", 28)
println(data._2)  // Arun
```

✅ Options (Some / None)

```
def safeDivide(a: Int, b: Int): Option[Int] =
  if (b == 0) None else Some(a / b)

println(safeDivide(10, 2))  // Some(5)
println(safeDivide(10, 0))  // None
```

🟧 2. Scala OOP Basics

✅ Classes

```
class Person(val name: String, val age: Int)

val p = new Person("Dani", 30)
println(p.name)
```

✅ Objects (Singleton)

```
object Utils {
  def greet(name: String) = s"Hello, $name"
}

println(Utils.greet("Scala"))
```

✅ Case Classes

```
case class Employee(id: Int, name: String)

val e = Employee(1, "Arun")
println(e.name)
```

✅ Companion Objects

```
class Counter(val value: Int)

object Counter {
  def apply(v: Int): Counter = new Counter(v)
}

val c = Counter(10)
println(c.value)
```

🟨 3. Functional Programming Essentials

✅ map

```
val nums = List(1,2,3)
val doubled = nums.map(_ * 2)
```

✅ filter

```
val evens = nums.filter(_ % 2 == 0)
```

✅ flatMap

```
val expanded = nums.flatMap(n => List(n, n * 10))
```

✅ reduce

```
val sum = nums.reduce(_ + _)
```

✅ fold

```
val total = nums.fold(0)(_ + _)
```

✅ Anonymous Functions

```
val sq = (x: Int) => x * x
println(sq(5))
```

🟩 4. Error Handling

✅ Try / Success / Failure

```
import scala.util.{Try, Success, Failure}

val result = Try(10 / 0)

result match {
  case Success(v) => println(v)
  case Failure(e) => println("Error: " + e.getMessage)
}
```

✅ Either

```
def divide(a: Int, b: Int): Either[String, Int] =
  if (b == 0) Left("Cannot divide by zero")
  else Right(a / b)

println(divide(10, 2))
println(divide(10, 0))
```

✅ Pattern Matching

```
def describe(x: Any): String = x match {
  case 0 => "Zero"
  case s: String => s"String: $s"
  case _ => "Unknown"
}

println(describe("hello"))
```

🟦 5. Scala + Spark Integration

(Use Spark 3.5.1 + Scala 2.12.18 + Java 11)

🔹 SparkSession

```
val spark = SparkSession.builder()
  .appName("ScalaSpark")
  .master("local[*]")
  .getOrCreate()
```

🔹 UDFs

```
val toUpper = udf((s: String) => s.toUpperCase)
val df2 = df.withColumn("name_upper", toUpper($"name"))
```

🔹 Case class schema + Dataset

```
case class Person(id: Int, name: String)

val df = Seq((1, "Dani"), (2, "Arun")).toDF("id","name")
val ds = df.as[Person]  // Dataset[Person]

ds.show()
```

🟥 PHASE 4 — END-TO-END PROJECTS (CODE)

✅ 1. Batch ETL Project (CSV → Clean → Window → Delta)

```
val spark = SparkSession.builder()
  .appName("BatchETL")
  .master("local[*]")
  .getOrCreate()

import spark.implicits._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window

// 1. Read CSV
val df = spark.read.option("header","true").csv("input/sales.csv")

// 2. Validate schema
```

```
val cleaned = df
  .filter($"amount".isNotNull)
  .withColumn("amount", $"amount".cast("double"))

// 3. Window function
val w = Window.partitionBy("category").orderBy($"amount".desc)
val ranked = cleaned.withColumn("rank", dense_rank().over(w))

// 4. Aggregation
val agg =
cleaned.groupBy("category").agg(sum("amount").alias("total_sales"))

// 5. Write to Delta
agg.write.format("delta").mode("overwrite").save("output/delta/sales")
```

✅ 2. S3 → EMR → Redshift Pipeline (Pseudo Spark Code)

```
val df = spark.read.json("s3a://bucket/raw/events/")

val cleaned = df
  .withColumn("event_ts", to_timestamp($"timestamp"))
  .filter($"event_type".isNotNull)

// write temporary output to S3 (EMR intermediate)
cleaned.write.mode("overwrite").parquet("s3a://bucket/clean/events")

// load to Redshift
cleaned.write
  .format("jdbc")
  .option("url", redshiftUrl)
  .option("dbtable", "public.events")
  .option("user", "user")
  .option("password", "pwd")
  .save()
```

✅ 3. Data Quality Framework (DQ Checks)

```
// Null Check
val nullCount = df.filter($"id".isNull).count()

// Uniqueness Check
val total = df.count()
val distinct = df.select("id").distinct().count()

// Referential Check
val missingRef = orders.join(customers, orders("cust_id") ===
customers("id"), "left_anti")

// Regex Validation
val validPhone = df.filter($"phone".rlike("^[0-9]{10}$"))
```

✅ 4. Real-Time Pipeline (Kafka → Spark Streaming → Delta)

```
val spark = SparkSession.builder()
  .appName("KafkaStream")
  .master("local[*]")
  .getOrCreate()

import spark.implicits._
import org.apache.spark.sql.functions._

// Read Kafka Stream
val streamDF = spark.readStream
  .format("kafka")
```

```
  .option("kafka.bootstrap.servers", "localhost:9092")
  .option("subscribe", "events_topic")
  .load()

// Transform
val parsed = streamDF.selectExpr("CAST(value AS STRING)")
  .select(from_json($"value", schema).as("data"))
  .select("data.*")

// Write to Delta
parsed.writeStream
  .format("delta")
  .option("checkpointLocation", "chk/events")
  .outputMode("append")
  .start("delta/events")
  .awaitTermination()
```

🟪 PHASE 5 — TOOLING (Examples)

Git

```
git init
git add .
git commit -m "First commit"
git push origin main
```

---

sbt: build.sbt

```
scalaVersion := "2.12.18"

libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-core" % "3.5.1",
  "org.apache.spark" %% "spark-sql"  % "3.5.1"
```

```
)
```

Shell (Linux)

```
ls -l
cat file.txt
grep "error" logfile.log
awk -F',' '{print $1, $3}' data.csv
```

Databricks (Notebook cell)

```
val df = spark.read.json("/mnt/raw/data.json")
df.display()
```