🔹 1. Basics — Variables, Types, Printing

```
@main def basicsDemo(): Unit =
  // Immutable (val) vs Mutable (var)
  val name: String = "Kapil"
  var age: Int = 25

  println(s"My name is $name and I am $age years old")

  age += 1
  println(s"Next year, I will be $age")

  // Type inference
  val salary = 50000.50  // Double
  println(s"Salary: $salary")
```

🔹 2. Control Structures (if, match, loops)

```
@main def controlStructures(): Unit =
  val marks = 85

  val grade = if marks >= 90 then "A"
              else if marks >= 75 then "B"
              else "C"

  println(s"Grade: $grade")

  // Match expression (like switch)
  val day = "Monday"
  day match
    case "Monday" => println("Start of week")
    case "Friday" => println("Weekend coming!")
    case _ => println("Midweek")

  // For loop
  for i <- 1 to 5 do println(s"Value: $i")
```

```scala
  // While loop
  var count = 3
  while count > 0 do
    println(s"Countdown: $count")
    count -= 1
```

🔹 3. Functions

```scala
@main def functionDemo(): Unit =
  def add(a: Int, b: Int): Int = a + b
  def greet(name: String = "Guest"): String = s"Hello, $name!"

  println(add(10, 20))
  println(greet("Sneha"))
  println(greet()) // uses default
```

🔹 4. Higher-Order Functions & Lambdas

```scala
@main def hofDemo(): Unit =
  val numbers = List(1, 2, 3, 4, 5)

  val doubled = numbers.map(_ * 2)
  println(doubled)

  def operateOnList(list: List[Int], fn: Int => Int): List[Int] =
    list.map(fn)

  println(operateOnList(numbers, x => x * x))
```

🔹 5. Collections (List, Map, Set)

```scala
@main def collectionsDemo(): Unit =
  val fruits = List("Apple", "Banana", "Mango")
  val prices = Map("Apple" -> 100, "Banana" -> 40)
  val uniqueNums = Set(1, 2, 2, 3)

  println(fruits.head)
  println(prices("Apple"))
  println(uniqueNums)

  fruits.foreach(f => println(s"Fruit: $f"))
```

🔹 6. Case Classes and Pattern Matching

```
@main def caseClassDemo(): Unit =
  case class Employee(id: Int, name: String, dept: String)

  val emp1 = Employee(1, "Tharun", "IT")
  val emp2 = emp1.copy(name = "Sangeetha")

  println(emp1)
  println(emp2)

  emp2 match
    case Employee(_, "Sangeetha", _) => println("Found Sangeetha")
    case _ => println("Unknown Employee")
```

🔹 7. Options and Null Safety

```
@main def optionDemo(): Unit =
  val maybeValue: Option[Int] = Some(10)
  val noValue: Option[Int] = None

  println(maybeValue.getOrElse(0))
  println(noValue.getOrElse(0))

  // Pattern matching with Option
  maybeValue match
    case Some(v) => println(s"Value = $v")
    case None => println("No value found")
```

🔹 8. Classes, Objects, Inheritance

```
class Vehicle(val brand: String):
  def drive(): Unit = println(s"$brand vehicle is driving")

class Car(brand: String, val model: String) extends Vehicle(brand):
  override def drive(): Unit =
    println(s"Car $brand $model is driving fast!")

@main def inheritanceDemo(): Unit =
  val car = Car("Tesla", "Model S")
  car.drive()
```

🔹 9. Traits (Interfaces)

```
trait Printable:
  def printInfo(): Unit

class Student(val name: String, val marks: Int) extends Printable:
  def printInfo(): Unit =
    println(s"Student: $name, Marks: $marks")

@main def traitDemo(): Unit =
  val s = Student("Kapil", 95)
  s.printInfo()
```

🔹 10. Companion Objects & Apply Method

```
class Department(val id: Int, val name: String)

object Department:
  def apply(id: Int, name: String) = new Department(id, name)
  def printDept(d: Department): Unit = println(s"Dept: ${d.name}")

@main def companionDemo(): Unit =
  val d = Department(1, "Engineering")
  Department.printDept(d)
```

🔹 11. Enumerations (Scala 3 feature)

```
enum Role:
  case Admin, Manager, Analyst, Engineer

@main def enumDemo(): Unit =
  val role: Role = Role.Engineer
  println(s"Role: $role")
```

🔹 12. Collections Transformations (for Big Data)

```
@main def transformations(): Unit =
  val salaries = List(50000, 60000, 75000, 80000)

  val increased = salaries.map(_ * 1.10)  // 10% raise
  val filtered = increased.filter(_ > 65000)
  val total = filtered.sum

  println(s"Total after hike: $total")
```

🔹 13. For Comprehensions

```
@main def forComprehension(): Unit =
  val names = List("Kapil", "Sneha", "Tharun")
  val greetings = for name <- names yield s"Hello, $name!"
  greetings.foreach(println)
```

🔹 14. Exception Handling

```
@main def exceptionDemo(): Unit =
  try
    val result = 10 / 0
    println(result)
  catch
    case e: ArithmeticException => println("Division by zero!")
  finally
    println("Program ended.")
```

🔹 15. Working with Files (I/O)

```
import scala.io.Source
import java.io.PrintWriter

@main def fileIODemo(): Unit =
  val writer = PrintWriter("data.txt")
  writer.write("Scala 3 is awesome!\n")
  writer.close()

  val content = Source.fromFile("data.txt").getLines().mkString("\n")
  println(content)
```

🔹 16. Futures (Asynchronous Programming)

```
import scala.concurrent.*
import ExecutionContext.Implicits.global
import scala.concurrent.duration.*

@main def futureDemo(): Unit =
  val f1 = Future { Thread.sleep(1000); 100 }
  val f2 = Future { Thread.sleep(500); 200 }

  val result = for
    x <- f1
    y <- f2
  yield x + y

  println(Await.result(result, 3.seconds))
```

🔹 17. Pattern Matching on Collections

```
@main def patternMatchCollections(): Unit =
  val list = List(1, 2, 3)

  list match
    case Nil => println("Empty list")
    case head :: tail => println(s"Head: $head, Tail: $tail")
```

🔹 18. Given/Using (Scala 3 Context Parameters)

```
trait Show[T]:
  def show(t: T): String

given Show[Int] with
  def show(t: Int) = s"Integer: $t"

def printValue[T](t: T)(using s: Show[T]): Unit =
  println(s.show(t))

@main def givenUsingDemo(): Unit =
  printValue(42)
```

🔹 19. Tuples & Destructuring

```
@main def tupleDemo(): Unit =
  val person = ("Kapil", 30, "Chennai")
  val (name, age, city) = person
  println(s"$name is $age years old from $city")
```

🔹 20. JSON / ETL Example (Spark-like)

```
import scala.util.parsing.json.JSON

@main def jsonDemo(): Unit =
  val jsonStr = """{"name":"Sneha","age":28,"city":"Chennai"}"""
  val parsed = JSON.parseFull(jsonStr)
  println(parsed)
```

🔹 21. Simple ETL Simulation

```
@main def etlDemo(): Unit =
  val raw = List(
    ("Kapil", "IT", 50000),
    ("Sneha", "HR", 60000),
    ("Tharun", "Finance", 70000)
  )

  val transformed = raw.map { case (n, d, s) => (n.toUpperCase, d, s *
1.10) }
  transformed.foreach(println)
```