# Scala JVM functional-OO language

## CONCEPTS

Scala 3 uses indentation for blocks (braces optional) and almost everything is an expression that returns a value.

**Package** namespace, top-level defs
**Trait / class / enum** types
**Method** def
**Expression** value

Prefer val and immutable collections; reach for var and mutable.* only locally.

## VALUES AND METHODS

`val x: Int = 1`
Immutable binding

`var n = 0`
Mutable variable

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

`def greet(name: String = "you") = s"Hi $name"`
Default parameter and interpolation

`(a: Int) ⇒ a + 1`
Anonymous function

`xs.map(_ * 2)`
Placeholder lambda

## VALUES AND METHODS (cont.)

`if x > 0 then "pos" else "neg"`
Conditional expression

`@main def run() = println("hi")`
Program entry point

## CLASSES AND TRAITS

`class Dog(name: String)`
Class with constructor param

`case class Point(x: Int, y: Int)`
Immutable data class with equals

`p.copy(x = 5)`
Copy with a changed field

`trait Speaker: def speak(): String`
Trait with abstract method

`class Dog extends Speaker, Runner`
Mix in several traits

`object Util: def one = 1`
Singleton object

`enum Color: case Red, Green, Blue`
Simple enum

`enum Opt: case No; case Yes(v: Int)`
Enum with parameters (ADT)

## CLASSES AND TRAITS (cont.)

`sealed trait Shape`
Closed hierarchy, exhaustive match

`extension (s: String) def shout = s.toUpperCase`
Extension method

## PATTERN MATCHING

`x match case 1 ⇒ "one"; case _ ⇒ "other"`
Match a value

`case Point(0, y) ⇒`
Destructure a case class

`case n if n > 0 ⇒`
Guard condition

`case h :: t ⇒`
List head and tail

`case Some(v) ⇒ ; case None ⇒`
Match an Option

## COLLECTIONS

`List(1, 2, 3)`
Immutable list

`Map("a" → 1)`
Immutable map

`xs.map(f)`
Transform each element

`xs.filter(p)`
Keep matching elements

## COLLECTIONS (cont.)

`xs.flatMap(f)`
Map and flatten

`xs.foldLeft(0)(_ + _)`
Fold with seed

`xs.sortBy(_.age)`
Sort by key

`xs.groupBy(_.k)`
Group into a Map

`m.getOrElse(k, 0)`
Lookup with default

## OPTIONS, EITHER, FOR

`Option(x)`
Wrap, null becomes None

`opt.map(f).getOrElse(d)`
Transform with default

`Right(v) / Left(err)`
Either success or failure

`for x ← xs yield f(x)`
For comprehension to a new collection

`for x ← xs; y ← ys if x < y yield (x, y)`
Nested loops with filter

`for a ← optA; b ← optB yield a + b`
Combine Options or Eithers

`for x ← xs do println(x)`
Loop for side effects

`s"$a and ${a + 1}"`
String interpolation

## SCALA-CLI

`scala-cli run <file>.scala`
Compile and run

`scala-cli repl`
Interactive shell

`scala-cli test .`
Run tests

`scala-cli setup-ide .`
Generate IDE project

`//> using dep org::name:1.0`
Add a dependency in file

## RECIPES

### Script with a dependency

```
//> using scala 3.3.4
//> using dep com.lihaoyi::os-lib:0.10.0
@main def run() =
  println(os.pwd)
```

### Modelling with an ADT

```
enum Expr:
  case Num(n: Int)
  case Add(a: Expr, b: Expr)
def eval(e: Expr): Int = e match
  case Expr.Num(n)    ⇒ n
  case Expr.Add(a, b) ⇒ eval(a) + eval(b)
```

### Typeclass with given

```
trait Show[A]: def show(a: A): String
given Show[Int] = _.toString
def out[A](a: A)(using s: Show[A]) = s.show(a)
```