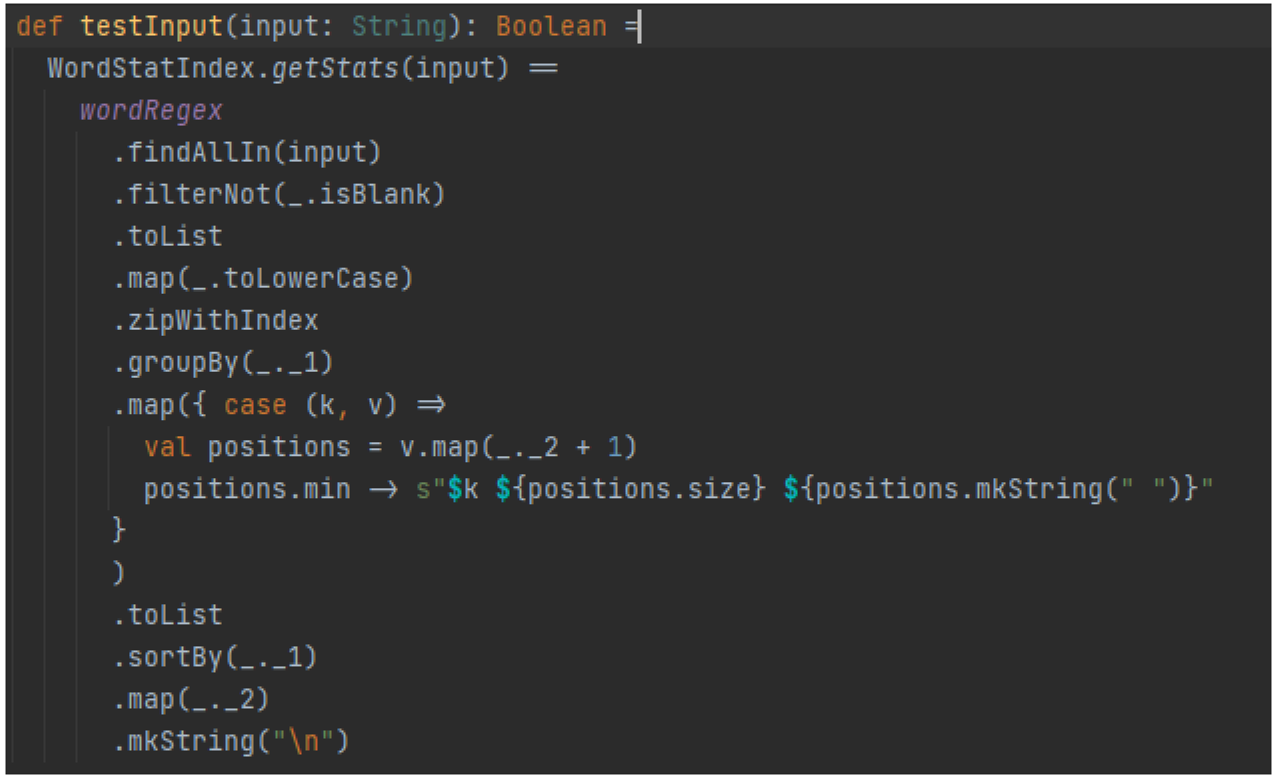
Scala Collections and Sugar

Пятая лекция

Scala Collections

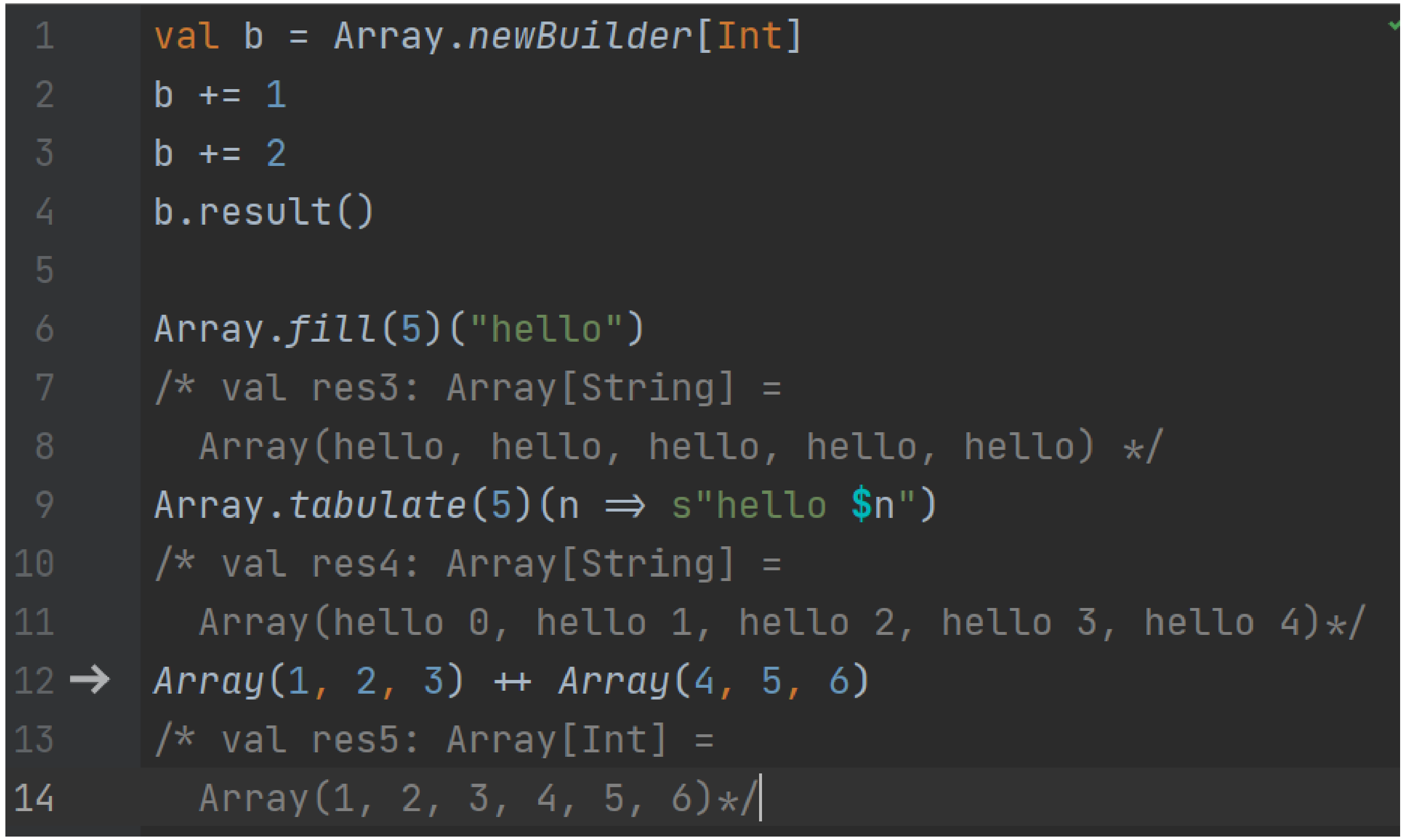


```scala
def testInput(input: String): Boolean =
  WordStatIndex.getStats(input) ==
    wordRegex
      .findAllIn(input)
      .filterNot(_.isBlank)
      .toList
      .map(_.toLowerCase)
      .zipWithIndex
      .groupBy(_._1)
      .map({ case (k, v) =>
        val positions = v.map(_._2 + 1)
        positions.min -> s"$k ${positions.size} ${positions.mkString(" ")}"
      }
      )
      .toList
      .sortBy(_._1)
      .map(_._2)
      .mkString("\n")
```

Scala Collections Operations

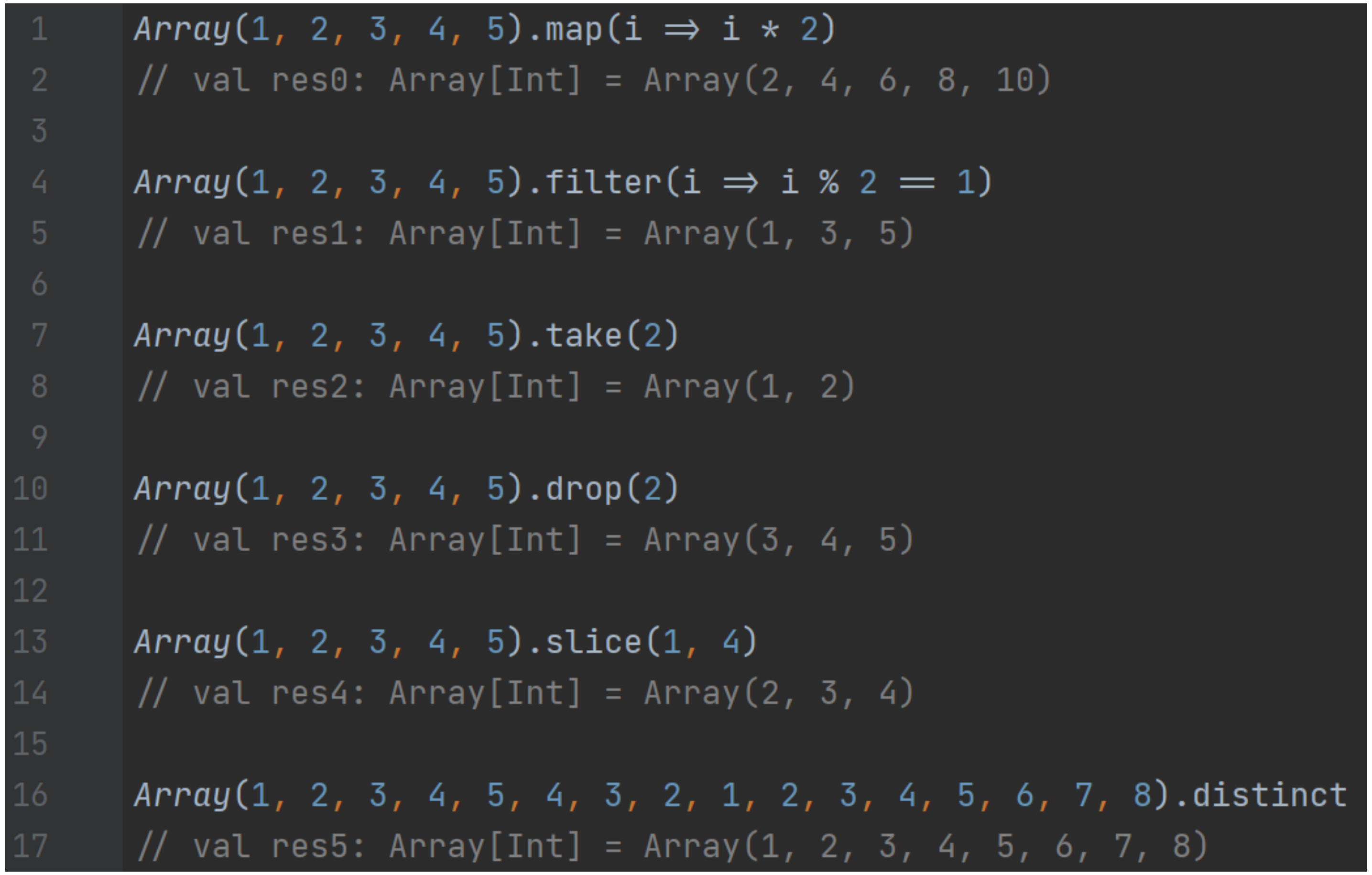
- Builders
- Factory Methods
- Transforms
- Queries
- Aggregations
- Combining Operations
- Converters
- Views

Builders/Factory methods



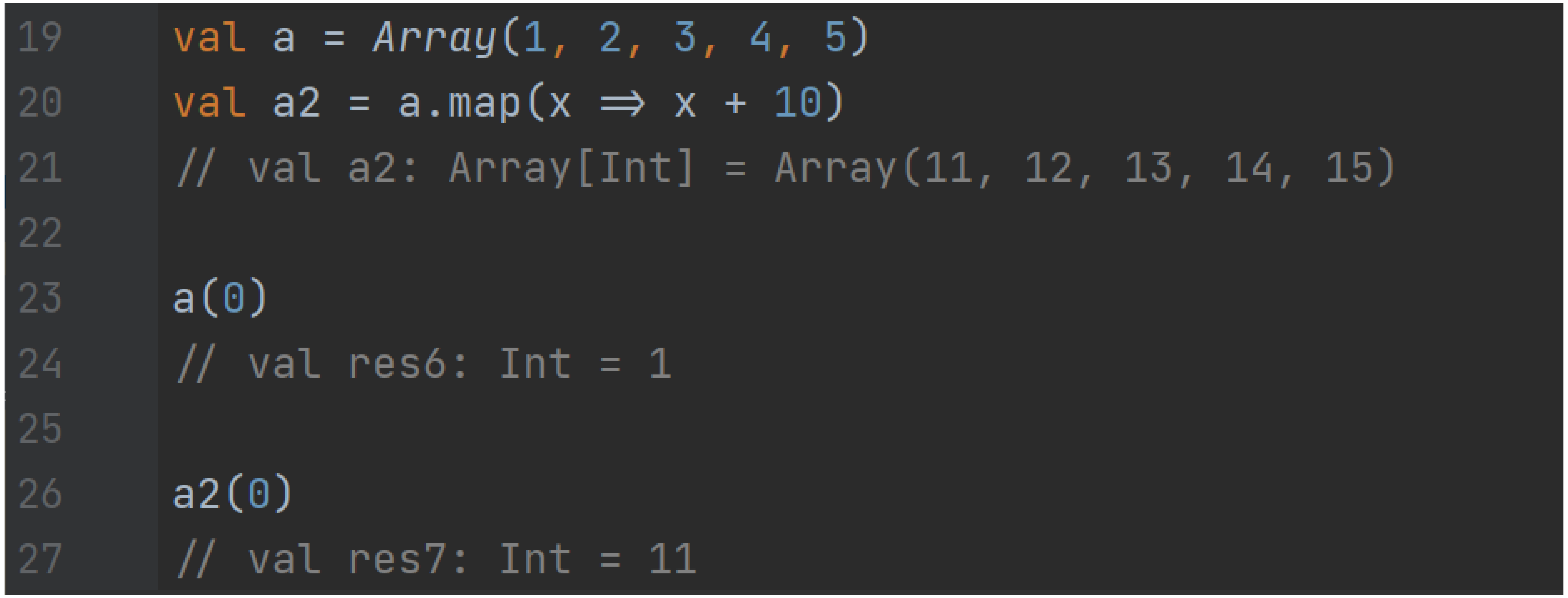
```
val b = Array.newBuilder[Int]
b += 1
b += 2
b.result()

Array.fill(5)("hello")
/* val res3: Array[String] =
  Array(hello, hello, hello, hello, hello) */
Array.tabulate(5)(n ⇒ s"hello $n")
/* val res4: Array[String] =
  Array(hello 0, hello 1, hello 2, hello 3, hello 4)*/
Array(1, 2, 3) ++ Array(4, 5, 6)
/* val res5: Array[Int] =
  Array(1, 2, 3, 4, 5, 6)*/
```

Transforms



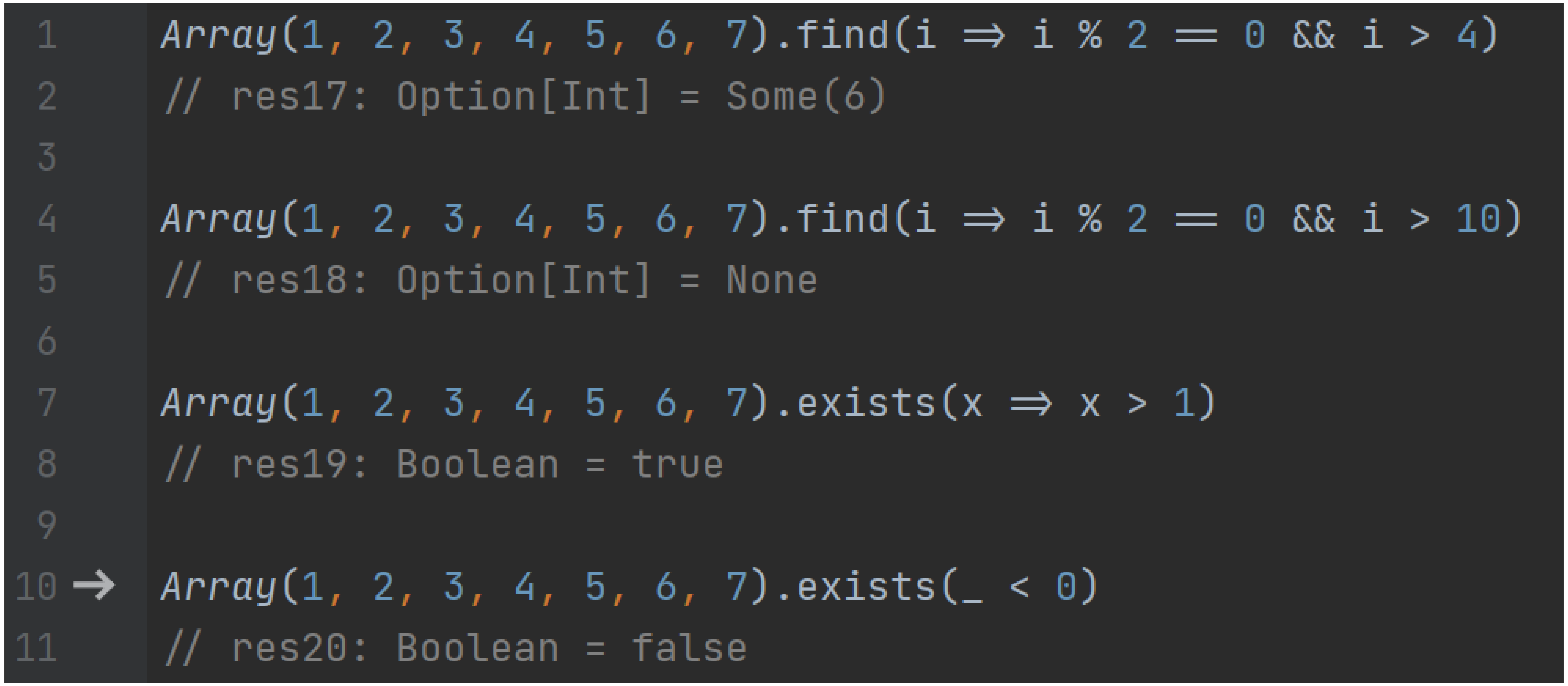
```scala
Array(1, 2, 3, 4, 5).map(i => i * 2)
// val res0: Array[Int] = Array(2, 4, 6, 8, 10)

Array(1, 2, 3, 4, 5).filter(i => i % 2 == 1)
// val res1: Array[Int] = Array(1, 3, 5)

Array(1, 2, 3, 4, 5).take(2)
// val res2: Array[Int] = Array(1, 2)

Array(1, 2, 3, 4, 5).drop(2)
// val res3: Array[Int] = Array(3, 4, 5)

Array(1, 2, 3, 4, 5).slice(1, 4)
// val res4: Array[Int] = Array(2, 3, 4)

Array(1, 2, 3, 4, 5, 4, 3, 2, 1, 2, 3, 4, 5, 6, 7, 8).distinct
// val res5: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8)
```

Transforms 2



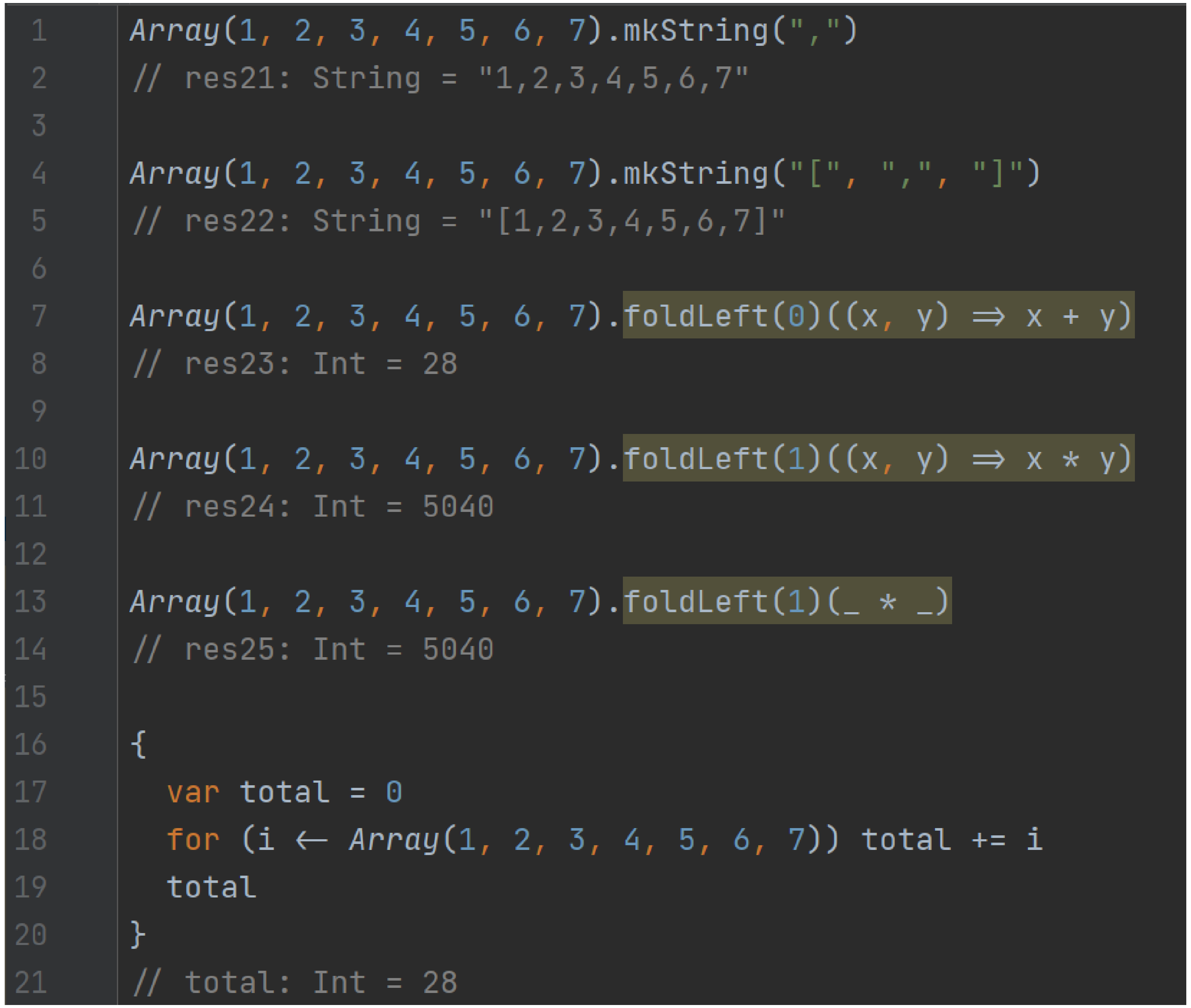
```scala
val a = Array(1, 2, 3, 4, 5)
val a2 = a.map(x ⇒ x + 10)
// val a2: Array[Int] = Array(11, 12, 13, 14, 15)

a(0)
// val res6: Int = 1

a2(0)
// val res7: Int = 11
```

Queries



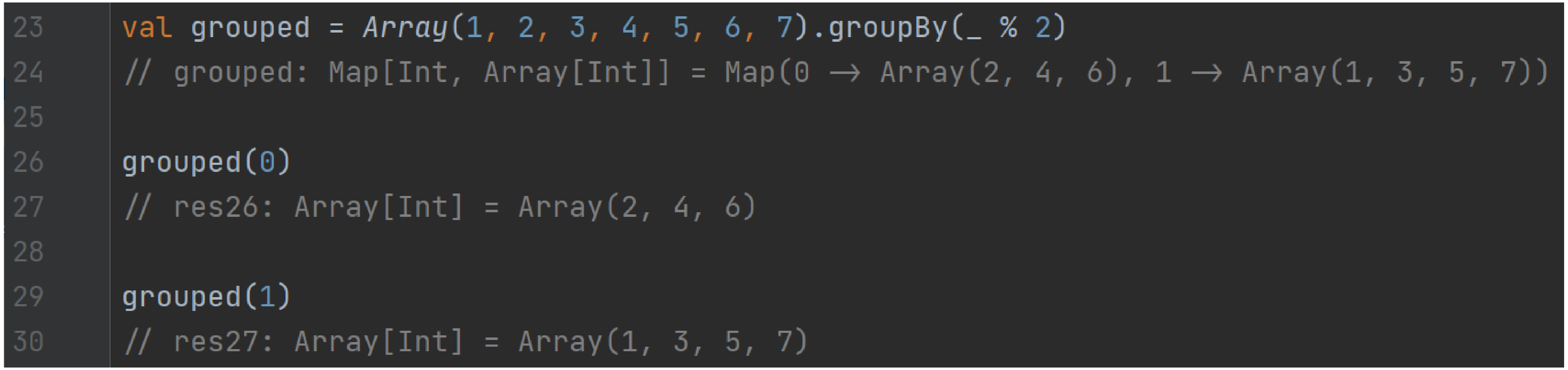
```scala
Array(1, 2, 3, 4, 5, 6, 7).find(i => i % 2 == 0 && i > 4)
// res17: Option[Int] = Some(6)

Array(1, 2, 3, 4, 5, 6, 7).find(i => i % 2 == 0 && i > 10)
// res18: Option[Int] = None

Array(1, 2, 3, 4, 5, 6, 7).exists(x => x > 1)
// res19: Boolean = true

Array(1, 2, 3, 4, 5, 6, 7).exists(_ < 0)
// res20: Boolean = false
```

Aggregations



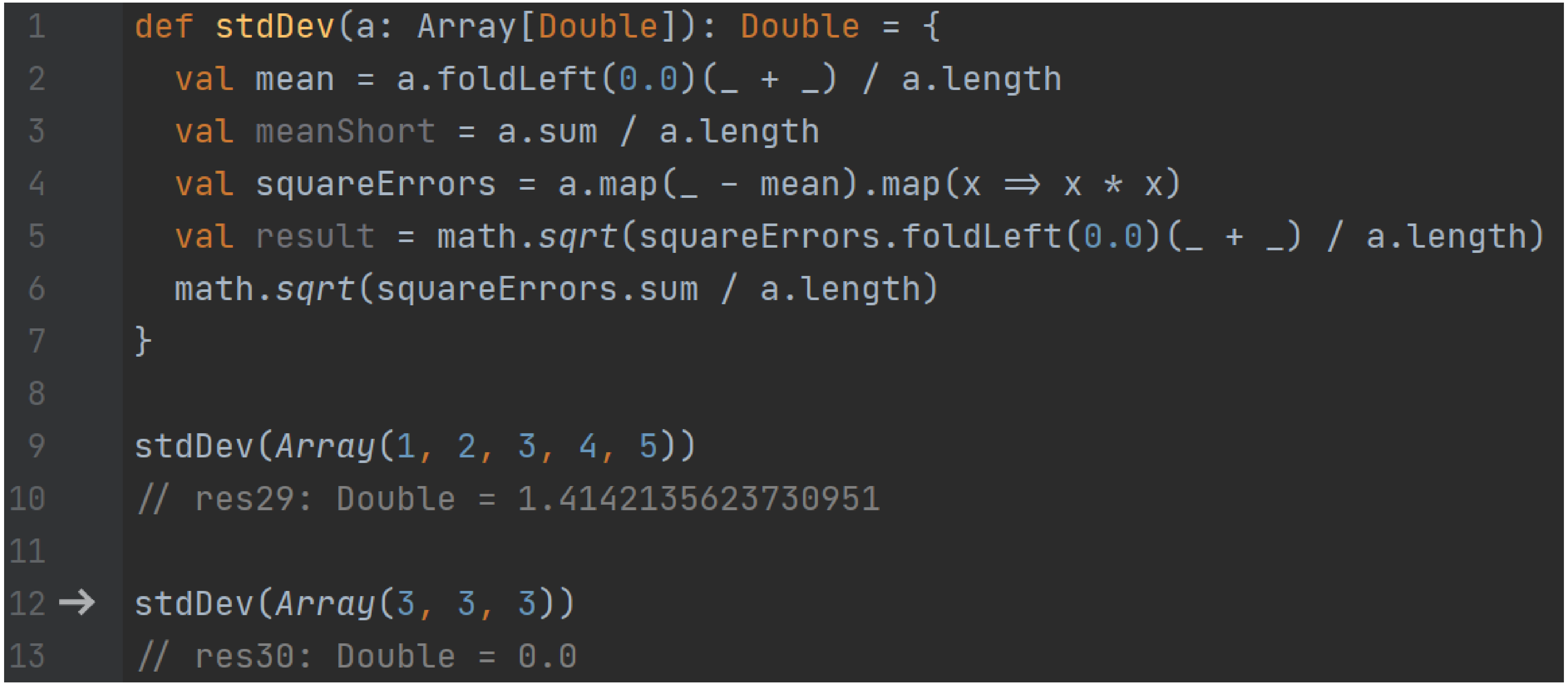
```scala
Array(1, 2, 3, 4, 5, 6, 7).mkString(",")
// res21: String = "1,2,3,4,5,6,7"

Array(1, 2, 3, 4, 5, 6, 7).mkString("[", ",", "]")
// res22: String = "[1,2,3,4,5,6,7]"

Array(1, 2, 3, 4, 5, 6, 7).foldLeft(0)((x, y) ⇒ x + y)
// res23: Int = 28

Array(1, 2, 3, 4, 5, 6, 7).foldLeft(1)((x, y) ⇒ x * y)
// res24: Int = 5040

Array(1, 2, 3, 4, 5, 6, 7).foldLeft(1)(_ * _)
// res25: Int = 5040

{
  var total = 0
  for (i ← Array(1, 2, 3, 4, 5, 6, 7)) total += i
  total
}
// total: Int = 28
```

Aggregations groupBy



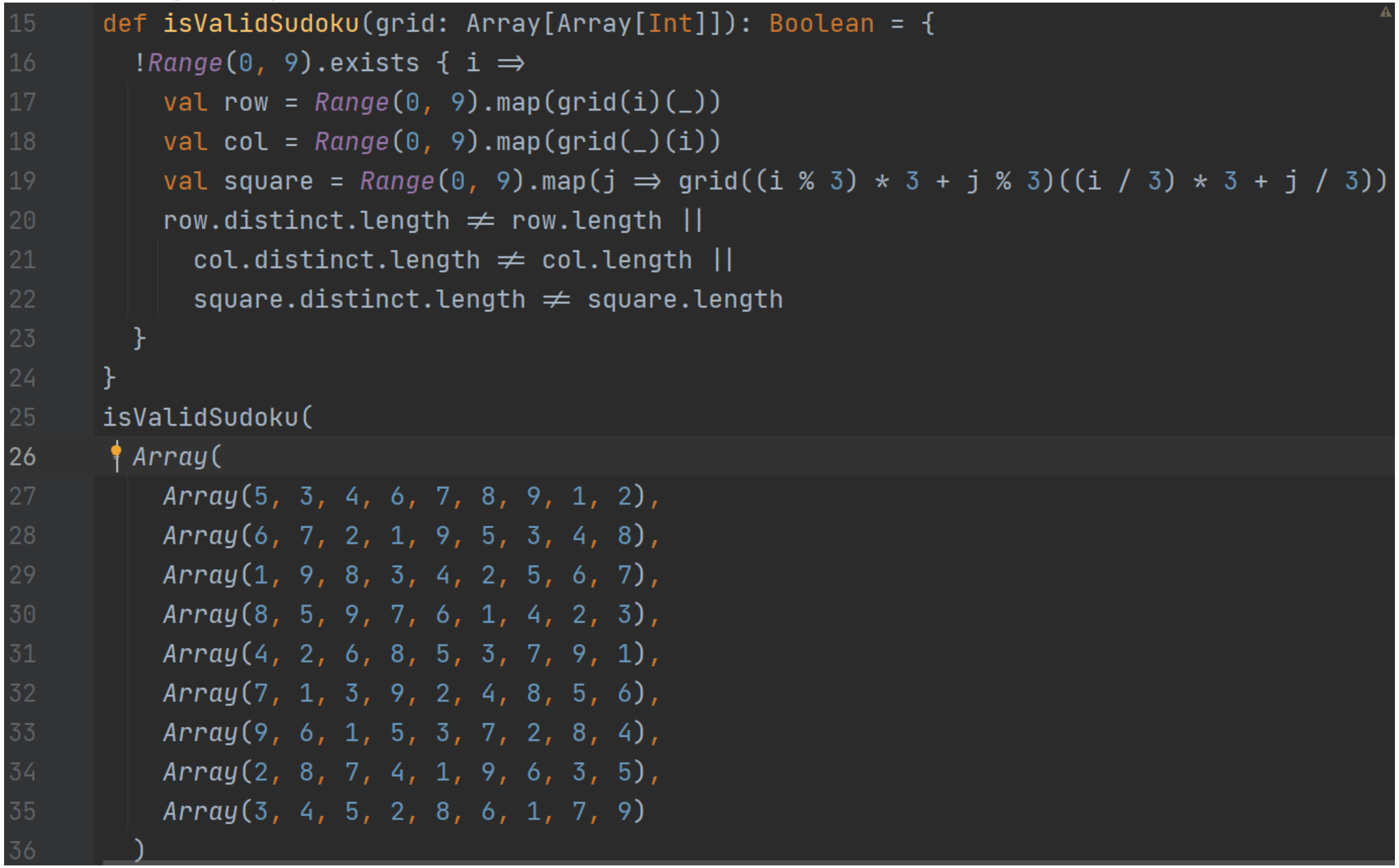
```scala
val grouped = Array(1, 2, 3, 4, 5, 6, 7).groupBy(_ % 2)
// grouped: Map[Int, Array[Int]] = Map(0 → Array(2, 4, 6), 1 → Array(1, 3, 5, 7))

grouped(0)
// res26: Array[Int] = Array(2, 4, 6)

grouped(1)
// res27: Array[Int] = Array(1, 3, 5, 7)
```

Chaining operations



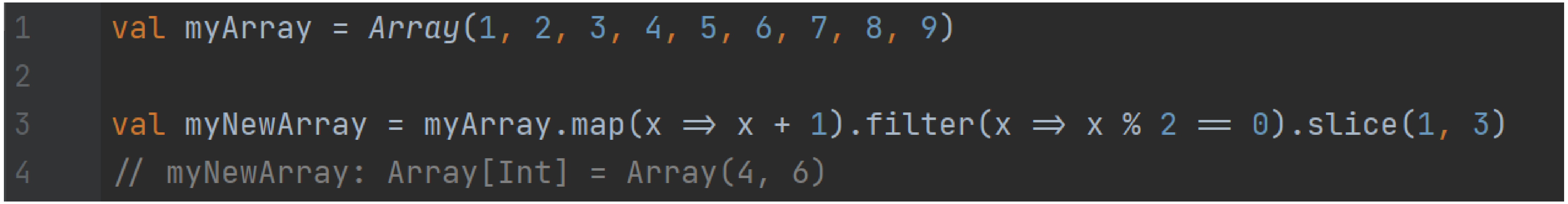
```scala
def stdDev(a: Array[Double]): Double = {
  val mean = a.foldLeft(0.0)(_ + _) / a.length
  val meanShort = a.sum / a.length
  val squareErrors = a.map(_ - mean).map(x => x * x)
  val result = math.sqrt(squareErrors.foldLeft(0.0)(_ + _) / a.length)
  math.sqrt(squareErrors.sum / a.length)
}

stdDev(Array(1, 2, 3, 4, 5))
// res29: Double = 1.4142135623730951

stdDev(Array(3, 3, 3))
// res30: Double = 0.0
```

Chaining operations 2

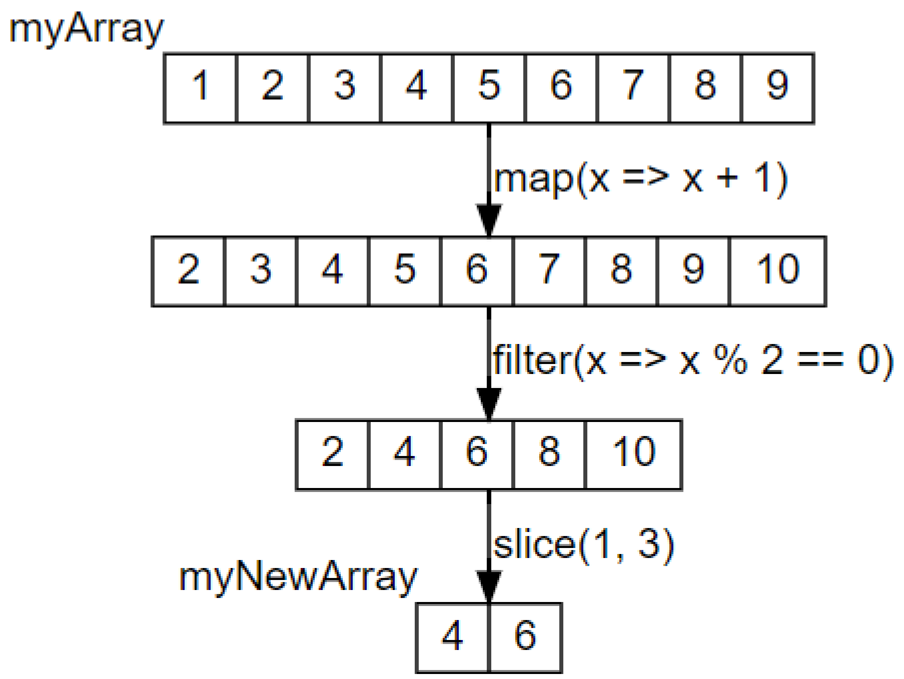


```scala
def isValidSudoku(grid: Array[Array[Int]]): Boolean = {
  !Range(0, 9).exists { i ⇒
    val row = Range(0, 9).map(grid(i)(_))
    val col = Range(0, 9).map(grid(_)(i))
    val square = Range(0, 9).map(j ⇒ grid((i % 3) * 3 + j % 3)((i / 3) * 3 + j / 3))
    row.distinct.length ≠ row.length ||
      col.distinct.length ≠ col.length ||
      square.distinct.length ≠ square.length
  }
}
isValidSudoku(
  Array(
    Array(5, 3, 4, 6, 7, 8, 9, 1, 2),
    Array(6, 7, 2, 1, 9, 5, 3, 4, 8),
    Array(1, 9, 8, 3, 4, 2, 5, 6, 7),
    Array(8, 5, 9, 7, 6, 1, 4, 2, 3),
    Array(4, 2, 6, 8, 5, 3, 7, 9, 1),
    Array(7, 1, 3, 9, 2, 4, 8, 5, 6),
    Array(9, 6, 1, 5, 3, 7, 2, 8, 4),
    Array(2, 8, 7, 4, 1, 9, 6, 3, 5),
    Array(3, 4, 5, 2, 8, 6, 1, 7, 9)
  )
```

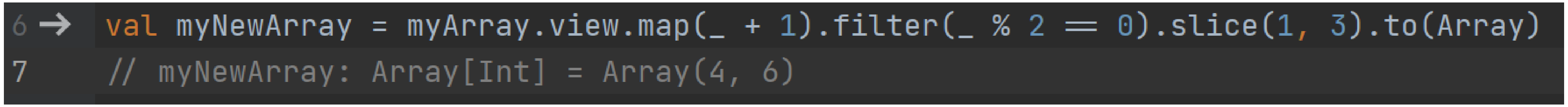
Views motivation



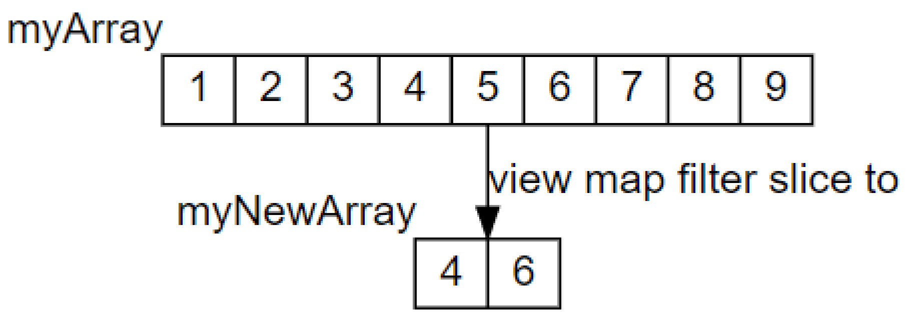
```scala
val myArray = Array(1, 2, 3, 4, 5, 6, 7, 8, 9)

val myNewArray = myArray.map(x ⇒ x + 1).filter(x ⇒ x % 2 == 0).slice(1, 3)
// myNewArray: Array[Int] = Array(4, 6)
```



Views



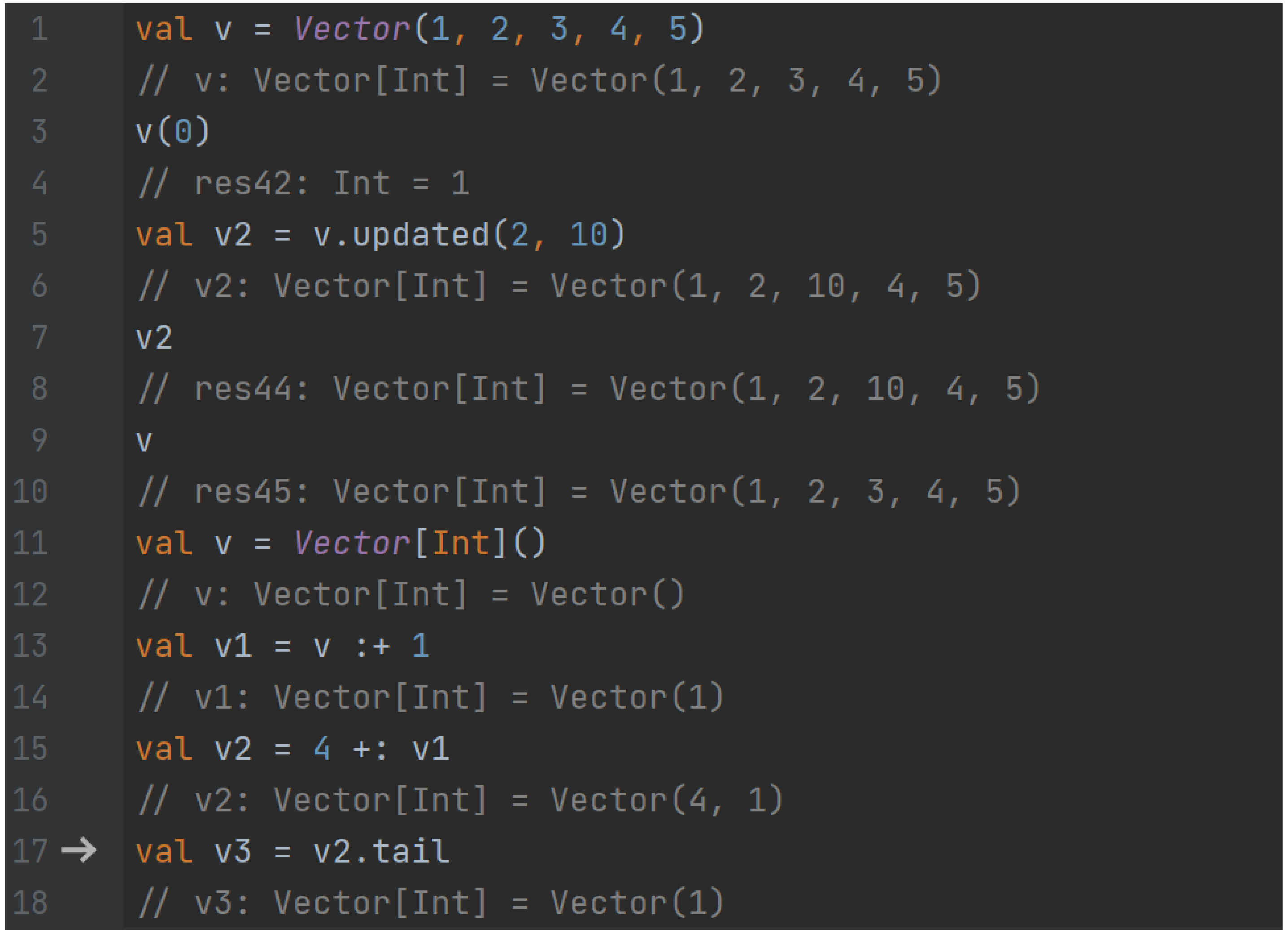
```
val myNewArray = myArray.view.map(_ + 1).filter(_ % 2 == 0).slice(1, 3).to(Array)
// myNewArray: Array[Int] = Array(4, 6)
```



Immutable Collections

- Vectors
- Structural Sharing
- Sets
- Maps
- List

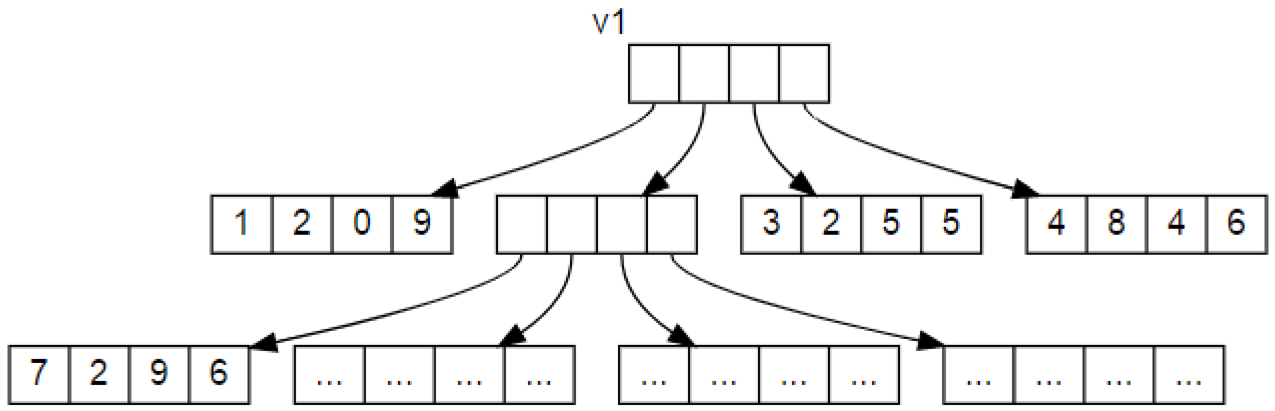
Immutable Vectors



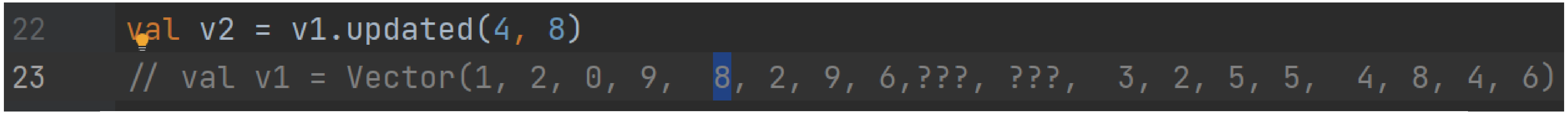
```scala
val v = Vector(1, 2, 3, 4, 5)
// v: Vector[Int] = Vector(1, 2, 3, 4, 5)
v(0)
// res42: Int = 1
val v2 = v.updated(2, 10)
// v2: Vector[Int] = Vector(1, 2, 10, 4, 5)
v2
// res44: Vector[Int] = Vector(1, 2, 10, 4, 5)
v
// res45: Vector[Int] = Vector(1, 2, 3, 4, 5)
val v = Vector[Int]()
// v: Vector[Int] = Vector()
val v1 = v :+ 1
// v1: Vector[Int] = Vector(1)
val v2 = 4 +: v1
// v2: Vector[Int] = Vector(4, 1)
val v3 = v2.tail
// v3: Vector[Int] = Vector(1)
```

Structural Sharing

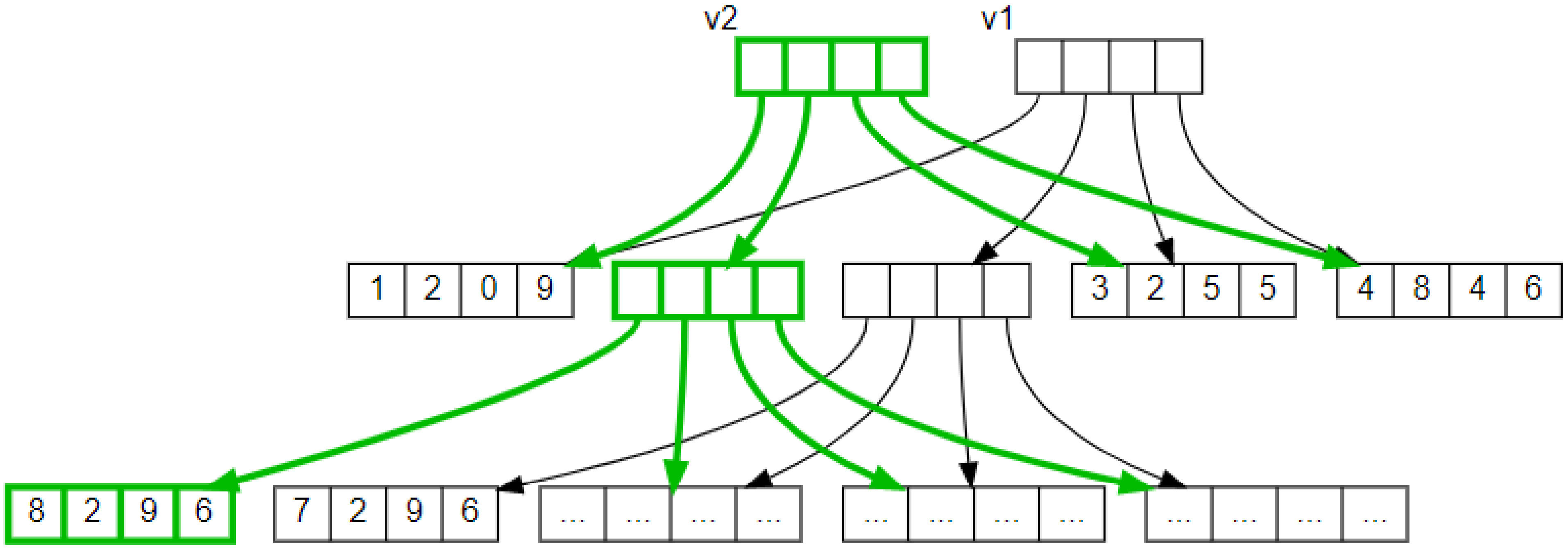
```scala
val v1 = Vector(1, 2, 0, 9,  7, 2, 9, 6,???, ???,  3, 2, 5, 5,  4, 8, 4, 6)
```



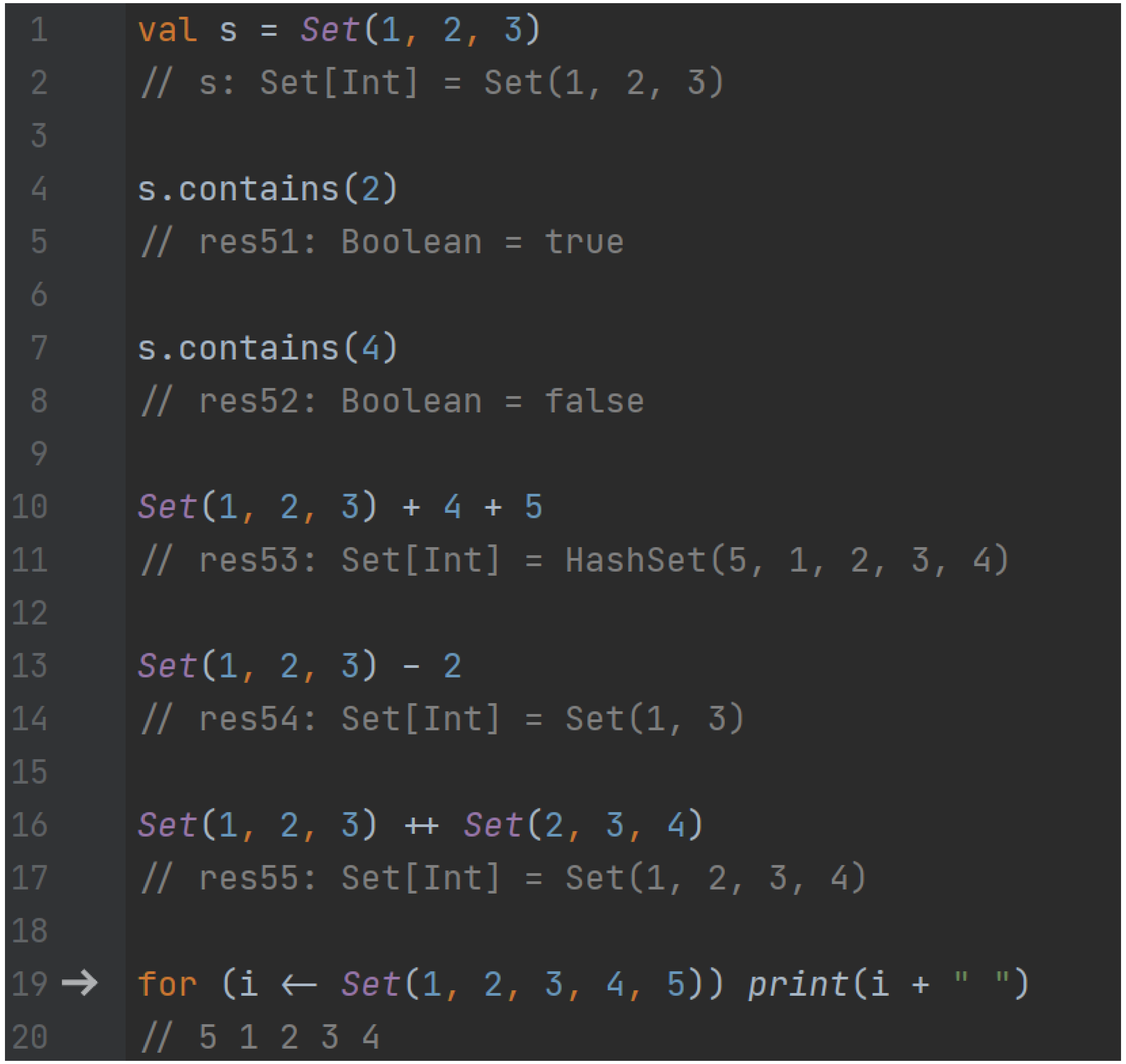
Structural Sharing 2



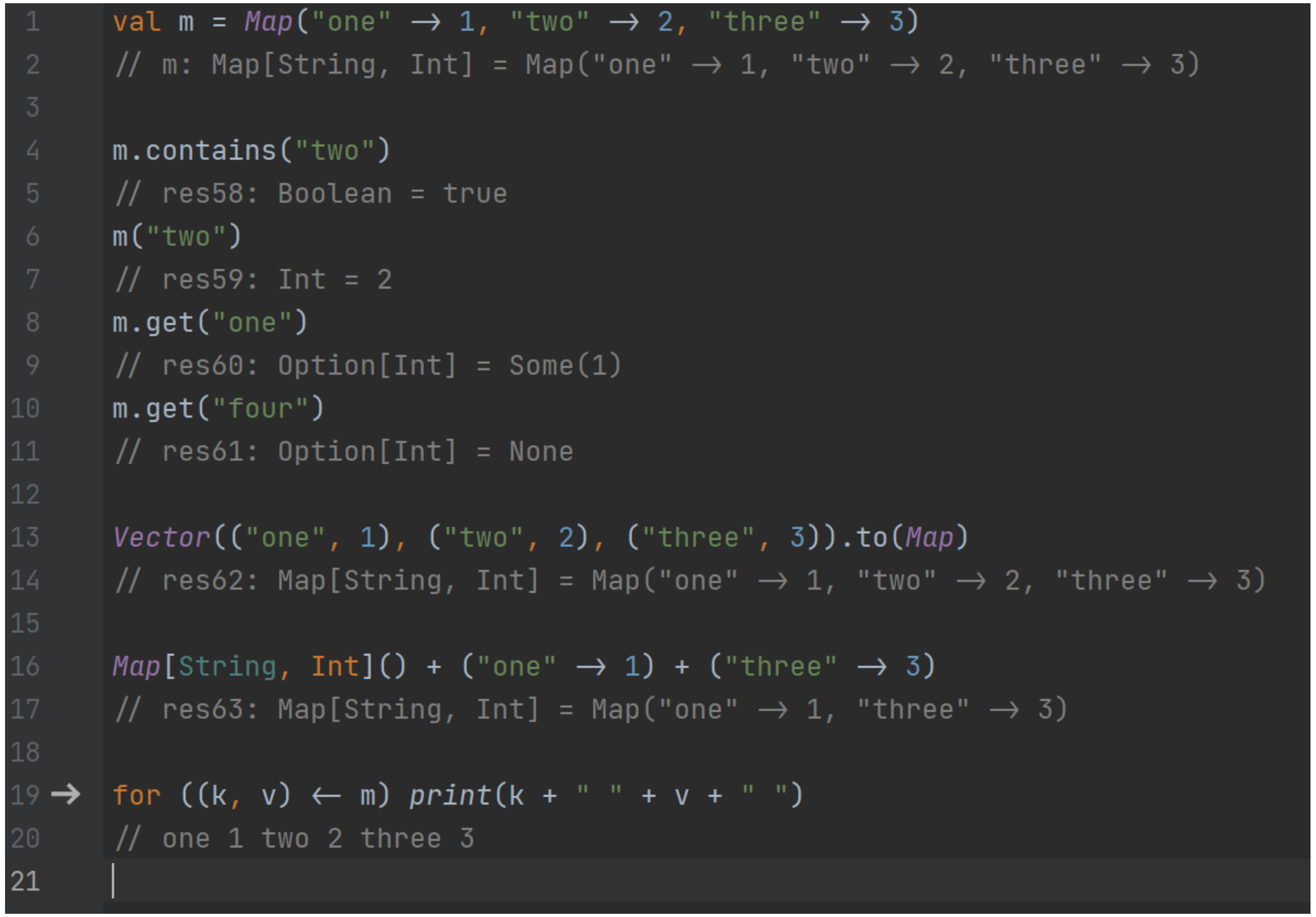
```
val v2 = v1.updated(4, 8)
// val v1 = Vector(1, 2, 0, 9,  8, 2, 9, 6,???, ???,  3, 2, 5, 5,  4, 8, 4, 6)
```



Sets

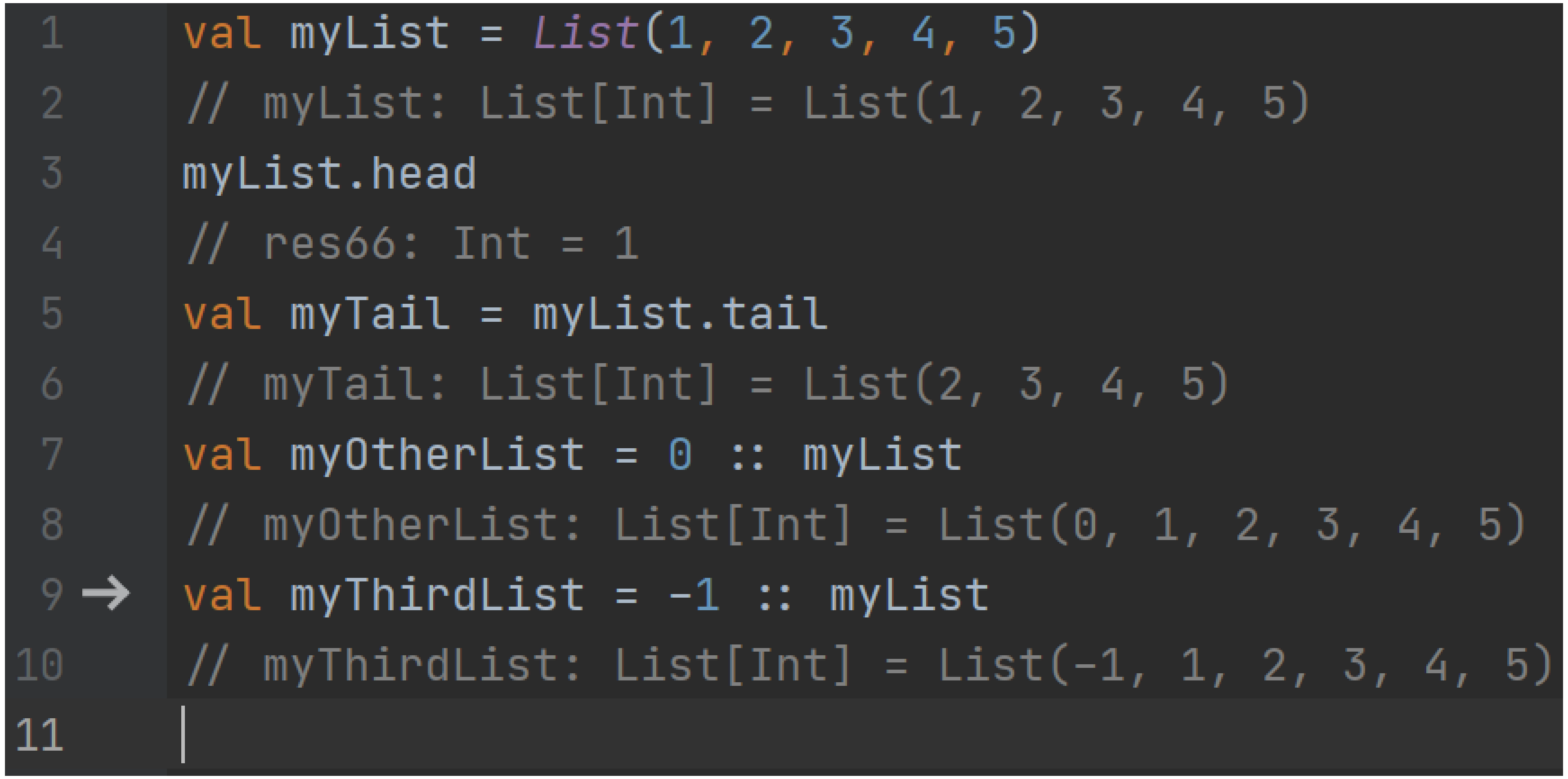

```scala
val s = Set(1, 2, 3)
// s: Set[Int] = Set(1, 2, 3)

s.contains(2)
// res51: Boolean = true

s.contains(4)
// res52: Boolean = false

Set(1, 2, 3) + 4 + 5
// res53: Set[Int] = HashSet(5, 1, 2, 3, 4)

Set(1, 2, 3) - 2
// res54: Set[Int] = Set(1, 3)

Set(1, 2, 3) ++ Set(2, 3, 4)
// res55: Set[Int] = Set(1, 2, 3, 4)

for (i ← Set(1, 2, 3, 4, 5)) print(i + " ")
// 5 1 2 3 4
```

Maps



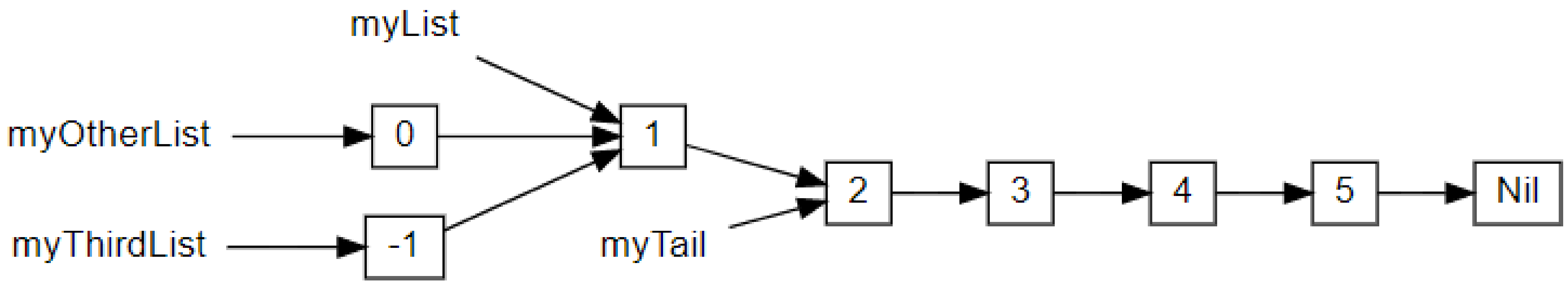
```scala
val m = Map("one" → 1, "two" → 2, "three" → 3)
// m: Map[String, Int] = Map("one" → 1, "two" → 2, "three" → 3)

m.contains("two")
// res58: Boolean = true
m("two")
// res59: Int = 2
m.get("one")
// res60: Option[Int] = Some(1)
m.get("four")
// res61: Option[Int] = None

Vector(("one", 1), ("two", 2), ("three", 3)).to(Map)
// res62: Map[String, Int] = Map("one" → 1, "two" → 2, "three" → 3)

Map[String, Int]() + ("one" → 1) + ("three" → 3)
// res63: Map[String, Int] = Map("one" → 1, "three" → 3)

for ((k, v) ← m) print(k + " " + v + " ")
// one 1 two 2 three 3

```

Lists



```scala
val myList = List(1, 2, 3, 4, 5)
// myList: List[Int] = List(1, 2, 3, 4, 5)
myList.head
// res66: Int = 1
val myTail = myList.tail
// myTail: List[Int] = List(2, 3, 4, 5)
val myOtherList = 0 :: myList
// myOtherList: List[Int] = List(0, 1, 2, 3, 4, 5)
val myThirdList = -1 :: myList
// myThirdList: List[Int] = List(-1, 1, 2, 3, 4, 5)
```

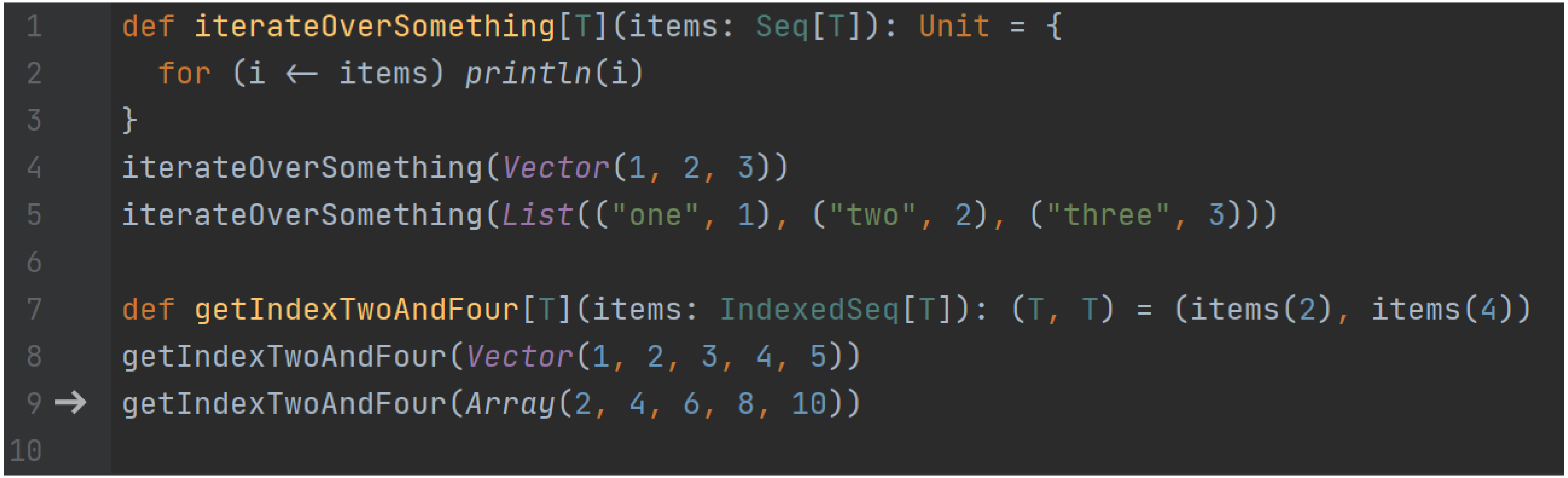
List Structural Sharing



Mutable Collections

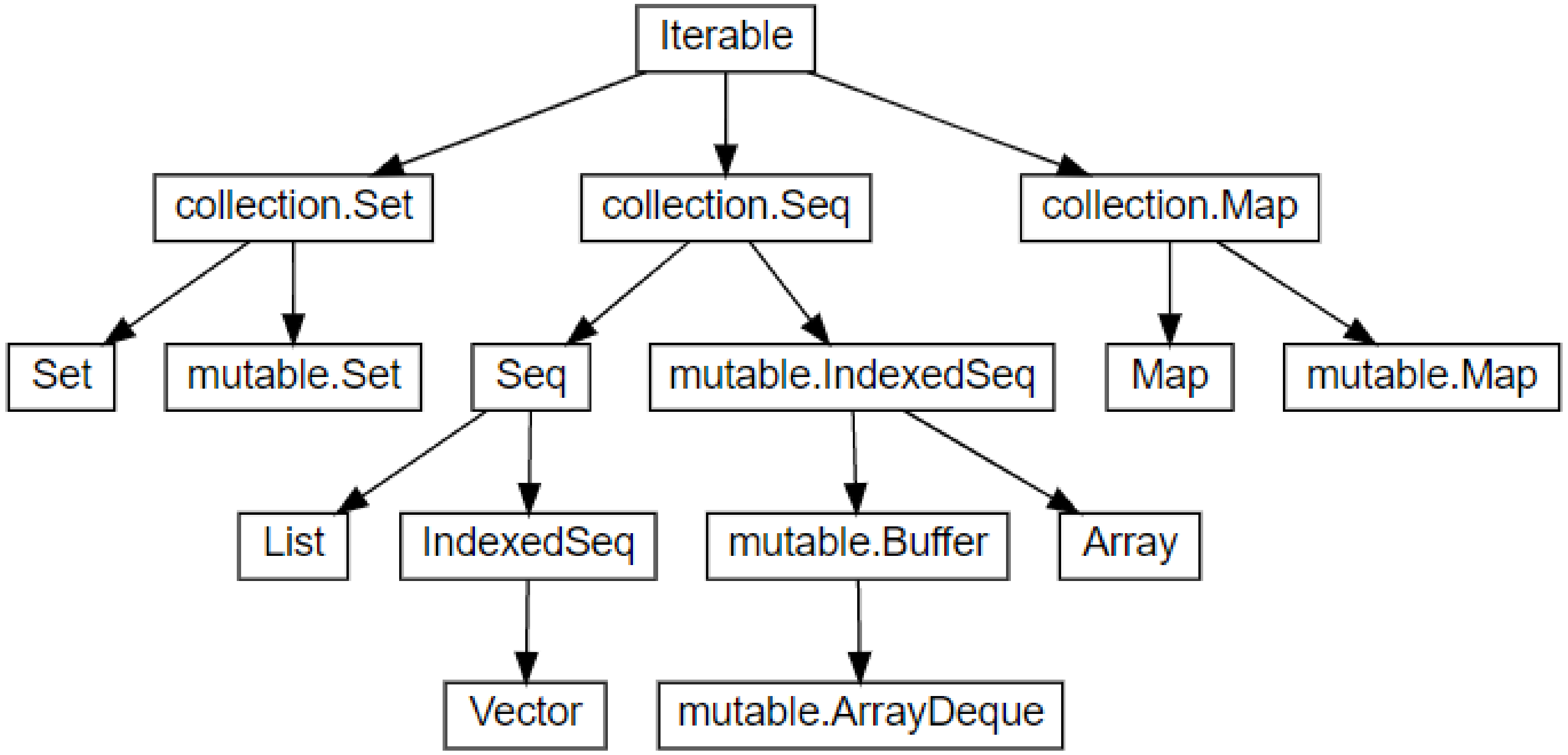
- ArrayDeque
- Set
- Map
- In-Place Operations

Common Traits



```scala
def iterateOverSomething[T](items: Seq[T]): Unit = {
  for (i ← items) println(i)
}
iterateOverSomething(Vector(1, 2, 3))
iterateOverSomething(List(("one", 1), ("two", 2), ("three", 3)))

def getIndexTwoAndFour[T](items: IndexedSeq[T]): (T, T) = (items(2), items(4))
getIndexTwoAndFour(Vector(1, 2, 3, 4, 5))
getIndexTwoAndFour(Array(2, 4, 6, 8, 10))
```

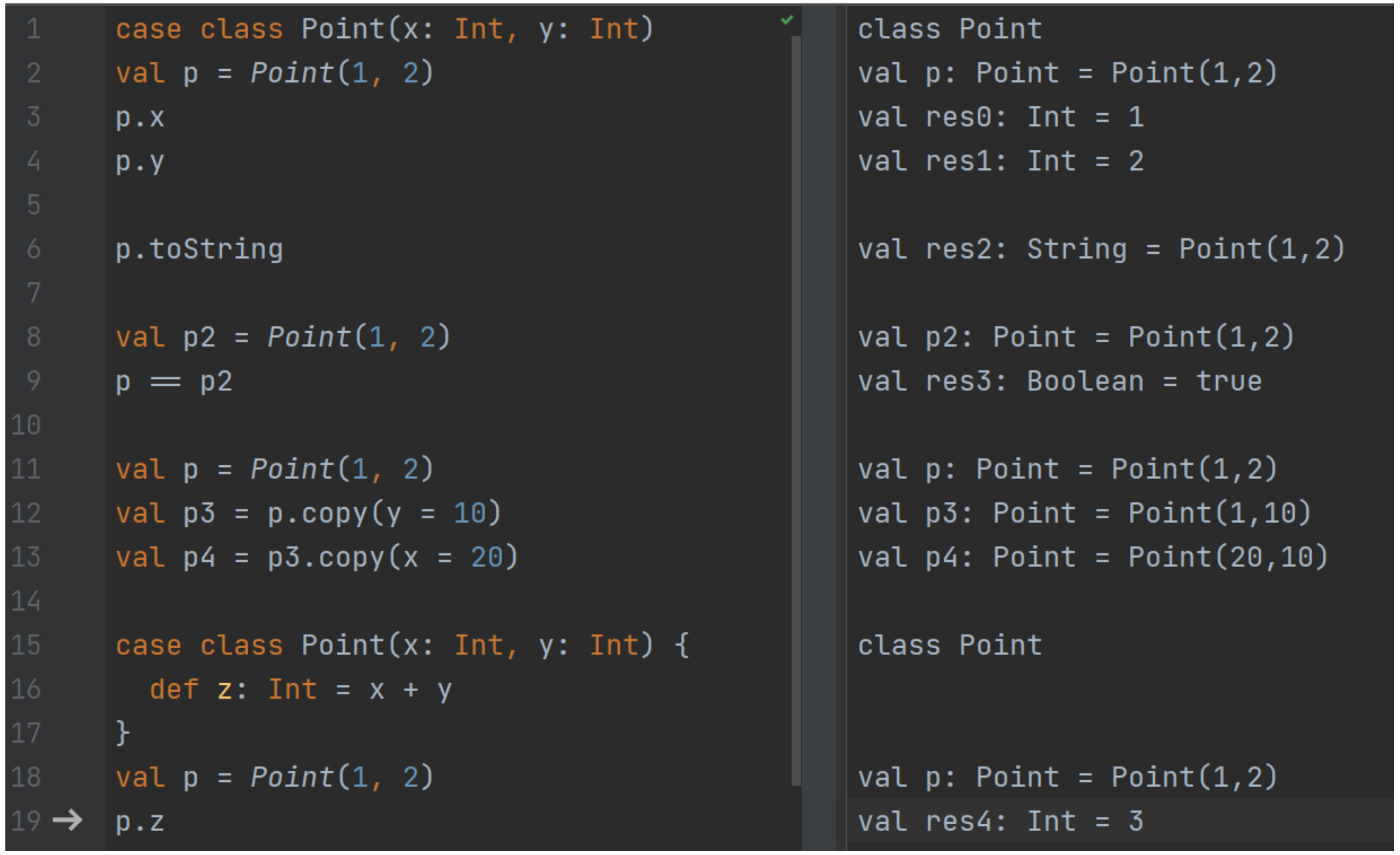
Collections hierarchy



Scala Sugar

- Case Classes and Sealed Traits
- Pattern Matching
- By-Name Parameters
- Implicit Parameters
- Typeclass Inference

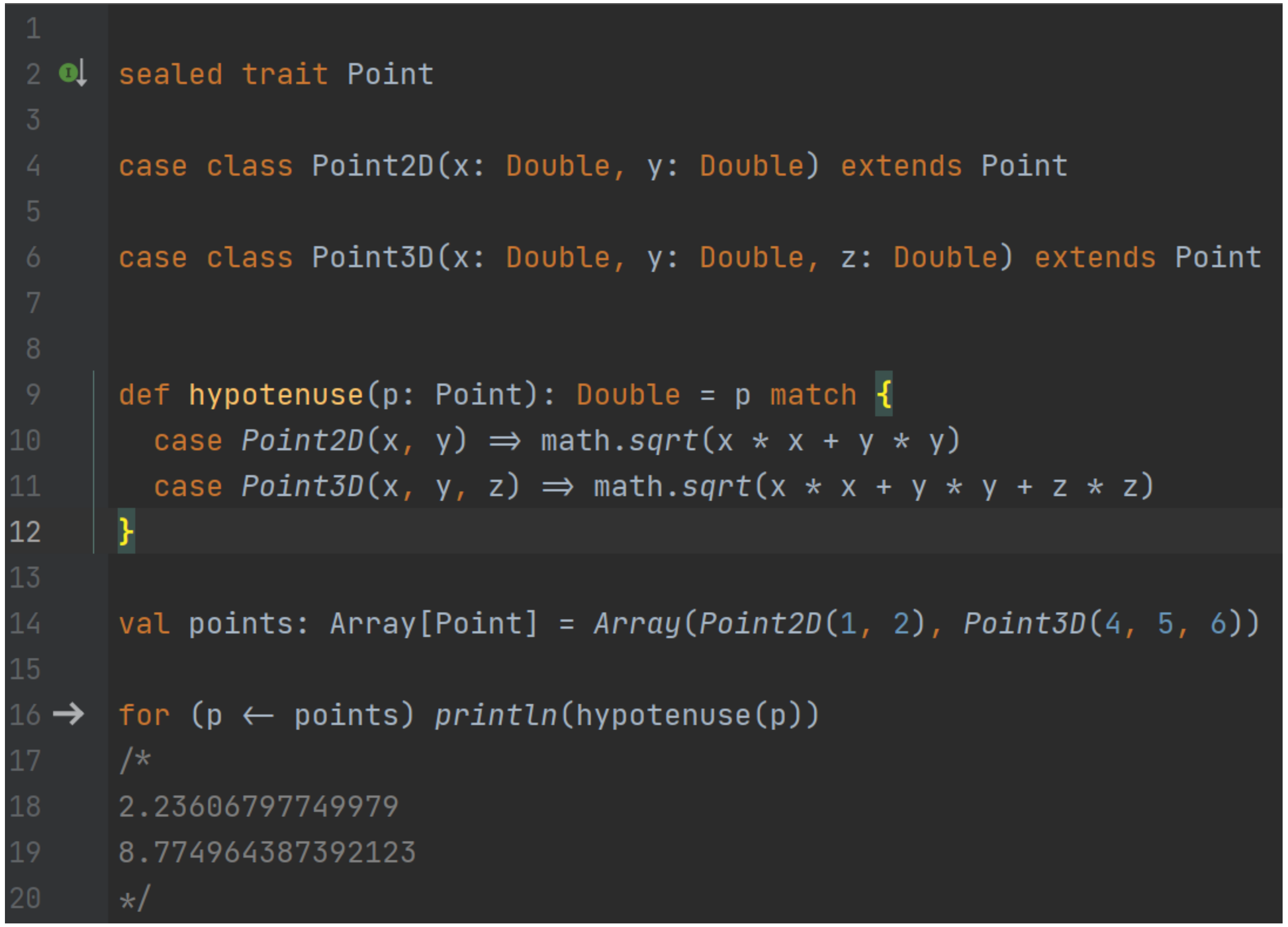
Case Classes



```scala
case class Point(x: Int, y: Int)
val p = Point(1, 2)
p.x
p.y

p.toString

val p2 = Point(1, 2)
p == p2

val p = Point(1, 2)
val p3 = p.copy(y = 10)
val p4 = p3.copy(x = 20)

case class Point(x: Int, y: Int) {
  def z: Int = x + y
}
val p = Point(1, 2)
p.z
```

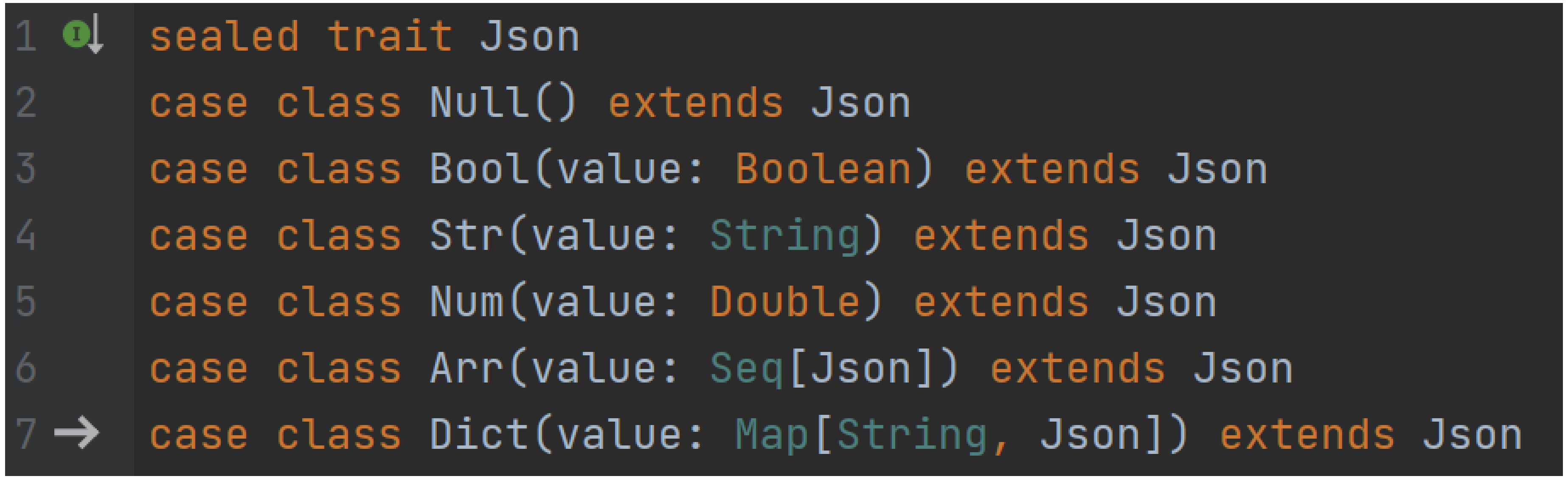
```
class Point
val p: Point = Point(1,2)
val res0: Int = 1
val res1: Int = 2

val res2: String = Point(1,2)

val p2: Point = Point(1,2)
val res3: Boolean = true

val p: Point = Point(1,2)
val p3: Point = Point(1,10)
val p4: Point = Point(20,10)

class Point


val p: Point = Point(1,2)
val res4: Int = 3
```

Sealed Traits



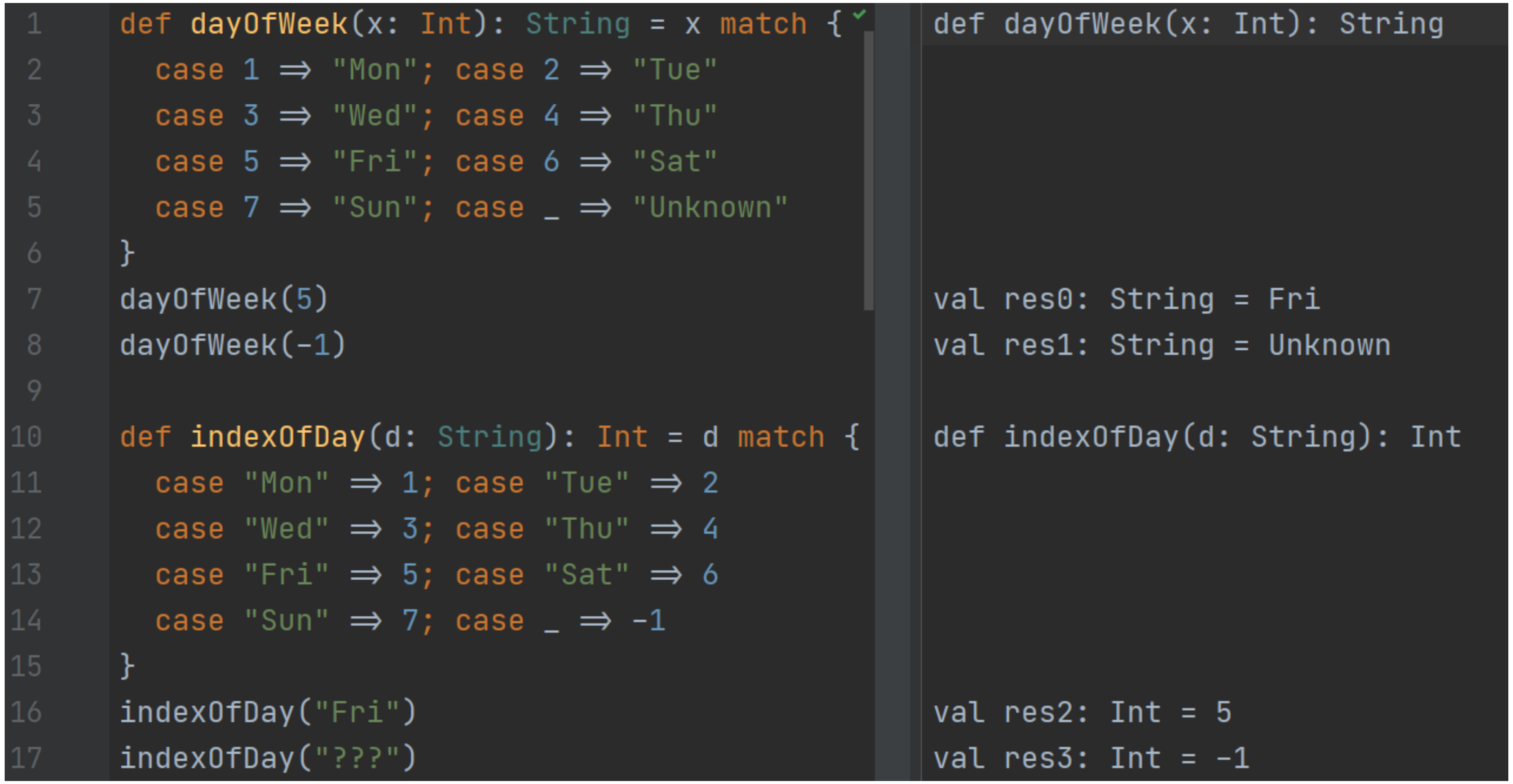
```scala

sealed trait Point

case class Point2D(x: Double, y: Double) extends Point

case class Point3D(x: Double, y: Double, z: Double) extends Point


def hypotenuse(p: Point): Double = p match {
  case Point2D(x, y) => math.sqrt(x * x + y * y)
  case Point3D(x, y, z) => math.sqrt(x * x + y * y + z * z)
}

val points: Array[Point] = Array(Point2D(1, 2), Point3D(4, 5, 6))

for (p <- points) println(hypotenuse(p))
/*
2.23606797749979
8.774964387392123
*/
```

Trait vs. Sealed Trait



```scala
sealed trait Json
case class Null() extends Json
case class Bool(value: Boolean) extends Json
case class Str(value: String) extends Json
case class Num(value: Double) extends Json
case class Arr(value: Seq[Json]) extends Json
case class Dict(value: Map[String, Json]) extends Json
```

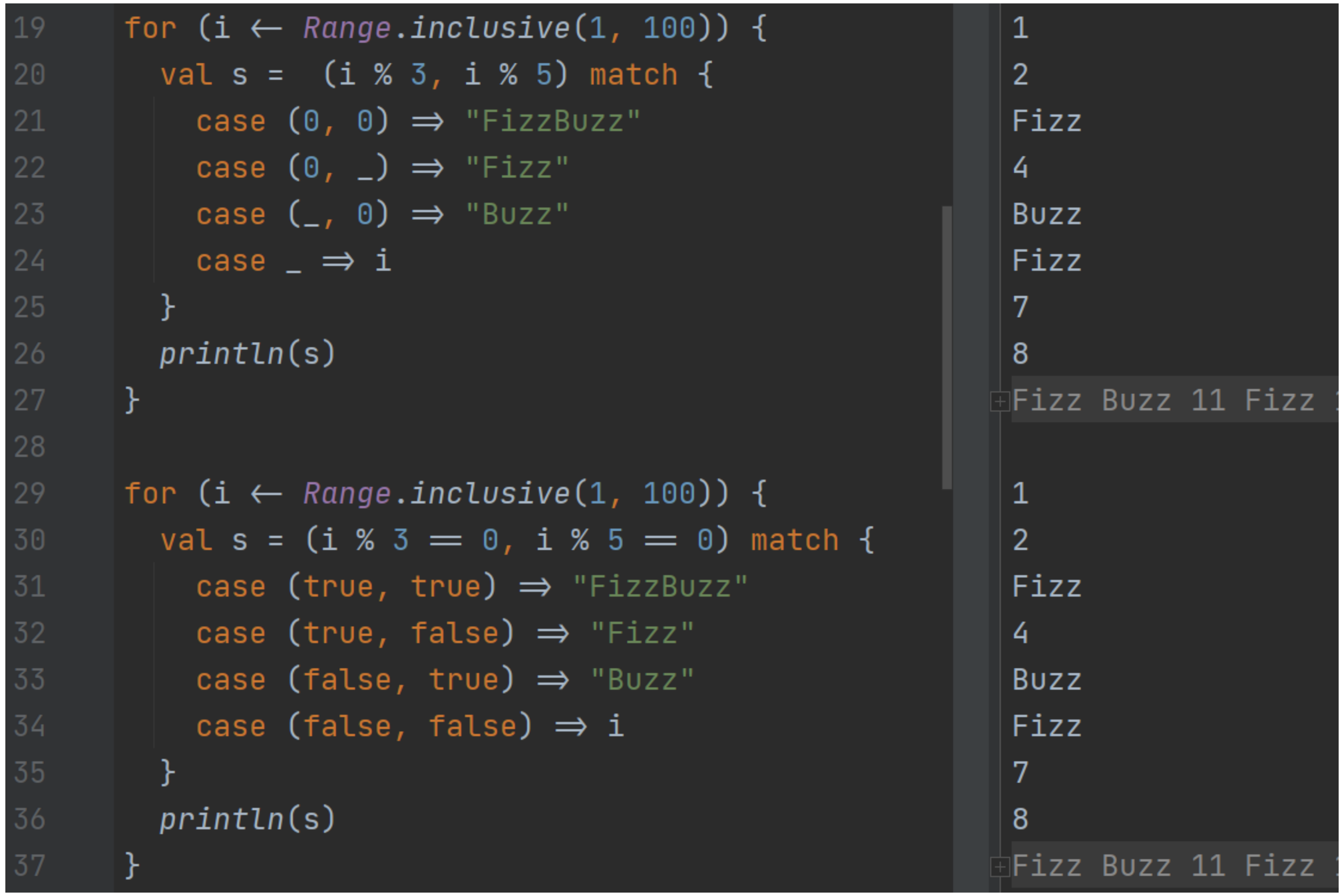
Pattern Matching 1



```scala
def dayOfWeek(x: Int): String = x match {
  case 1 ⇒ "Mon"; case 2 ⇒ "Tue"
  case 3 ⇒ "Wed"; case 4 ⇒ "Thu"
  case 5 ⇒ "Fri"; case 6 ⇒ "Sat"
  case 7 ⇒ "Sun"; case _ ⇒ "Unknown"
}
dayOfWeek(5)
dayOfWeek(-1)

def indexOfDay(d: String): Int = d match {
  case "Mon" ⇒ 1; case "Tue" ⇒ 2
  case "Wed" ⇒ 3; case "Thu" ⇒ 4
  case "Fri" ⇒ 5; case "Sat" ⇒ 6
  case "Sun" ⇒ 7; case _ ⇒ -1
}
indexOfDay("Fri")
indexOfDay("???")
```

```
def dayOfWeek(x: Int): String





val res0: String = Fri
val res1: String = Unknown

def indexOfDay(d: String): Int





val res2: Int = 5
val res3: Int = -1
```

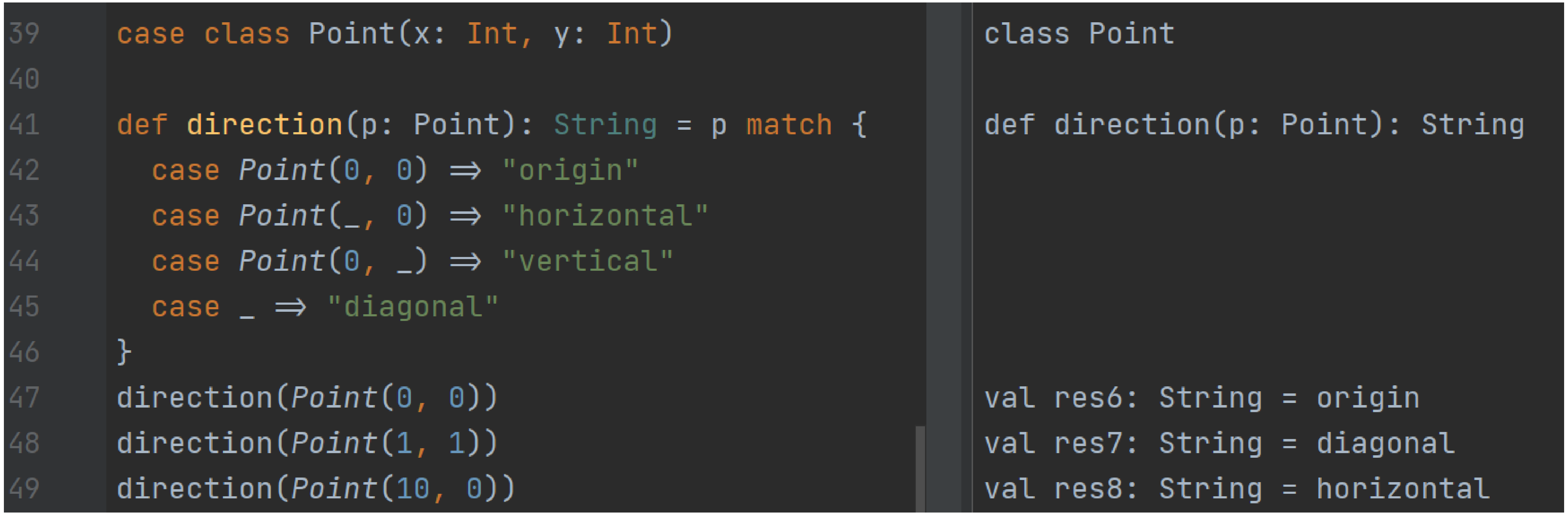
Pattern Matching 2



```
for (i ← Range.inclusive(1, 100)) {
  val s =  (i % 3, i % 5) match {
    case (0, 0) ⇒ "FizzBuzz"
    case (0, _) ⇒ "Fizz"
    case (_, 0) ⇒ "Buzz"
    case _ ⇒ i
  }
  println(s)
}

for (i ← Range.inclusive(1, 100)) {
  val s = (i % 3 == 0, i % 5 == 0) match {
    case (true, true) ⇒ "FizzBuzz"
    case (true, false) ⇒ "Fizz"
    case (false, true) ⇒ "Buzz"
    case (false, false) ⇒ i
  }
  println(s)
}
```

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz Buzz 11 Fizz

1
2
Fizz
4
Buzz
Fizz
7
8
Fizz Buzz 11 Fizz
```

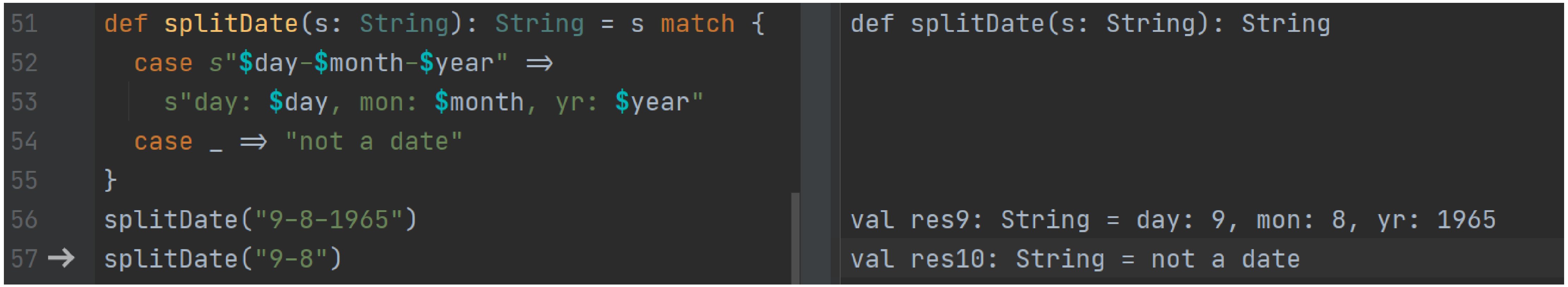
Pattern Matching Case Class



```scala
case class Point(x: Int, y: Int)

def direction(p: Point): String = p match {
  case Point(0, 0) ⇒ "origin"
  case Point(_, 0) ⇒ "horizontal"
  case Point(0, _) ⇒ "vertical"
  case _ ⇒ "diagonal"
}
direction(Point(0, 0))
direction(Point(1, 1))
direction(Point(10, 0))
```

```
class Point

def direction(p: Point): String





val res6: String = origin
val res7: String = diagonal
val res8: String = horizontal
```

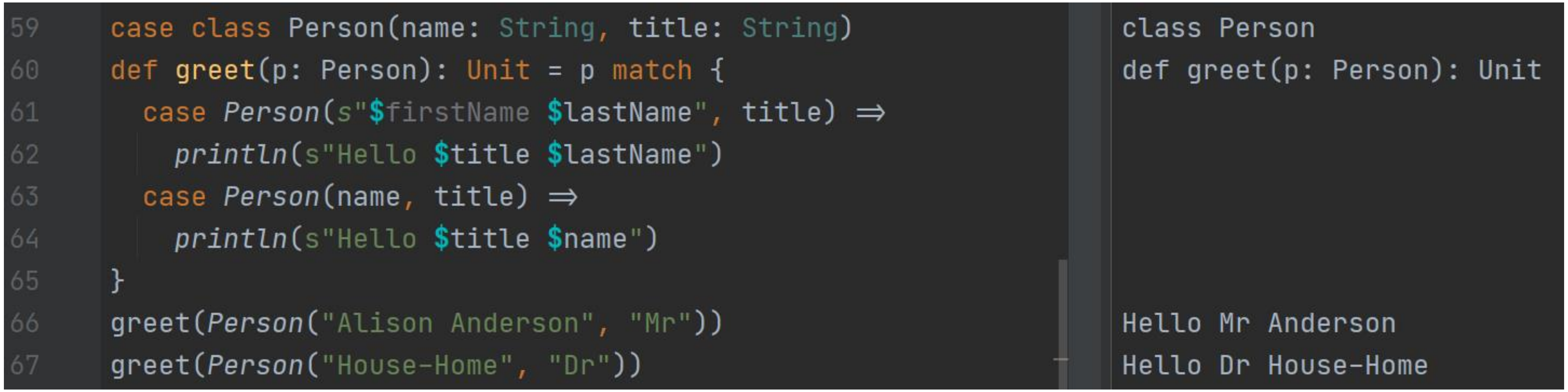
Pattern Matching String Pattern



```scala
def splitDate(s: String): String = s match {
  case s"$day-$month-$year" ⇒
    s"day: $day, mon: $month, yr: $year"
  case _ ⇒ "not a date"
}
splitDate("9-8-1965")
splitDate("9-8")
```

```
def splitDate(s: String): String




val res9: String = day: 9, mon: 8, yr: 1965
val res10: String = not a date
```

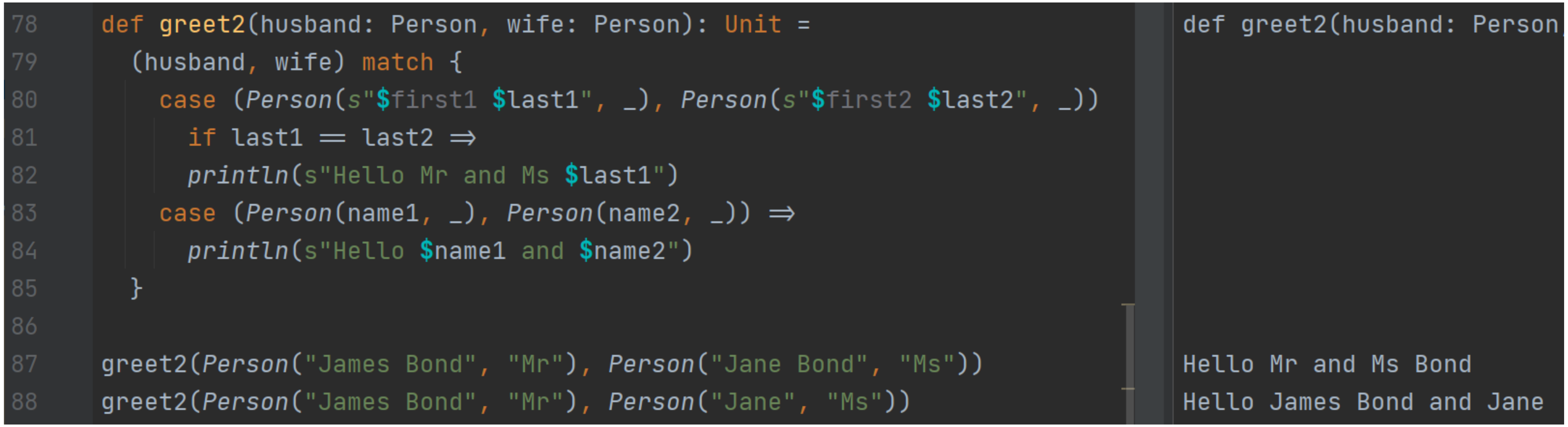
Nested Matches



```
case class Person(name: String, title: String)
def greet(p: Person): Unit = p match {
  case Person(s"$firstName $lastName", title) ⇒
    println(s"Hello $title $lastName")
  case Person(name, title) ⇒
    println(s"Hello $title $name")
}
greet(Person("Alison Anderson", "Mr"))
greet(Person("House-Home", "Dr"))
```

```
class Person
def greet(p: Person): Unit




Hello Mr Anderson
Hello Dr House-Home
```

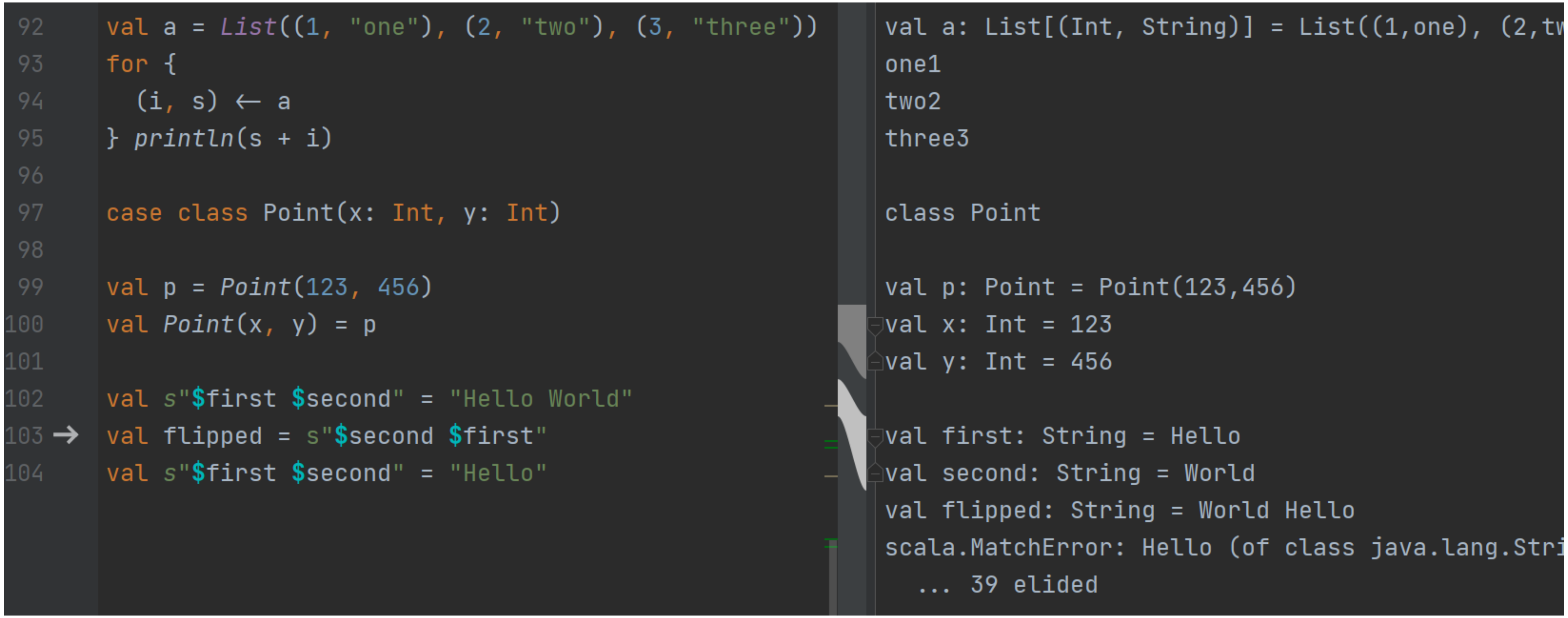
Nested Matches 2



```
def greet2(husband: Person, wife: Person): Unit =
  (husband, wife) match {
    case (Person(s"$first1 $last1", _), Person(s"$first2 $last2", _))
      if last1 == last2 =>
      println(s"Hello Mr and Ms $last1")
    case (Person(name1, _), Person(name2, _)) =>
      println(s"Hello $name1 and $name2")
  }

greet2(Person("James Bond", "Mr"), Person("Jane Bond", "Ms"))
greet2(Person("James Bond", "Mr"), Person("Jane", "Ms"))
```

```
def greet2(husband: Person
Hello Mr and Ms Bond
Hello James Bond and Jane
```

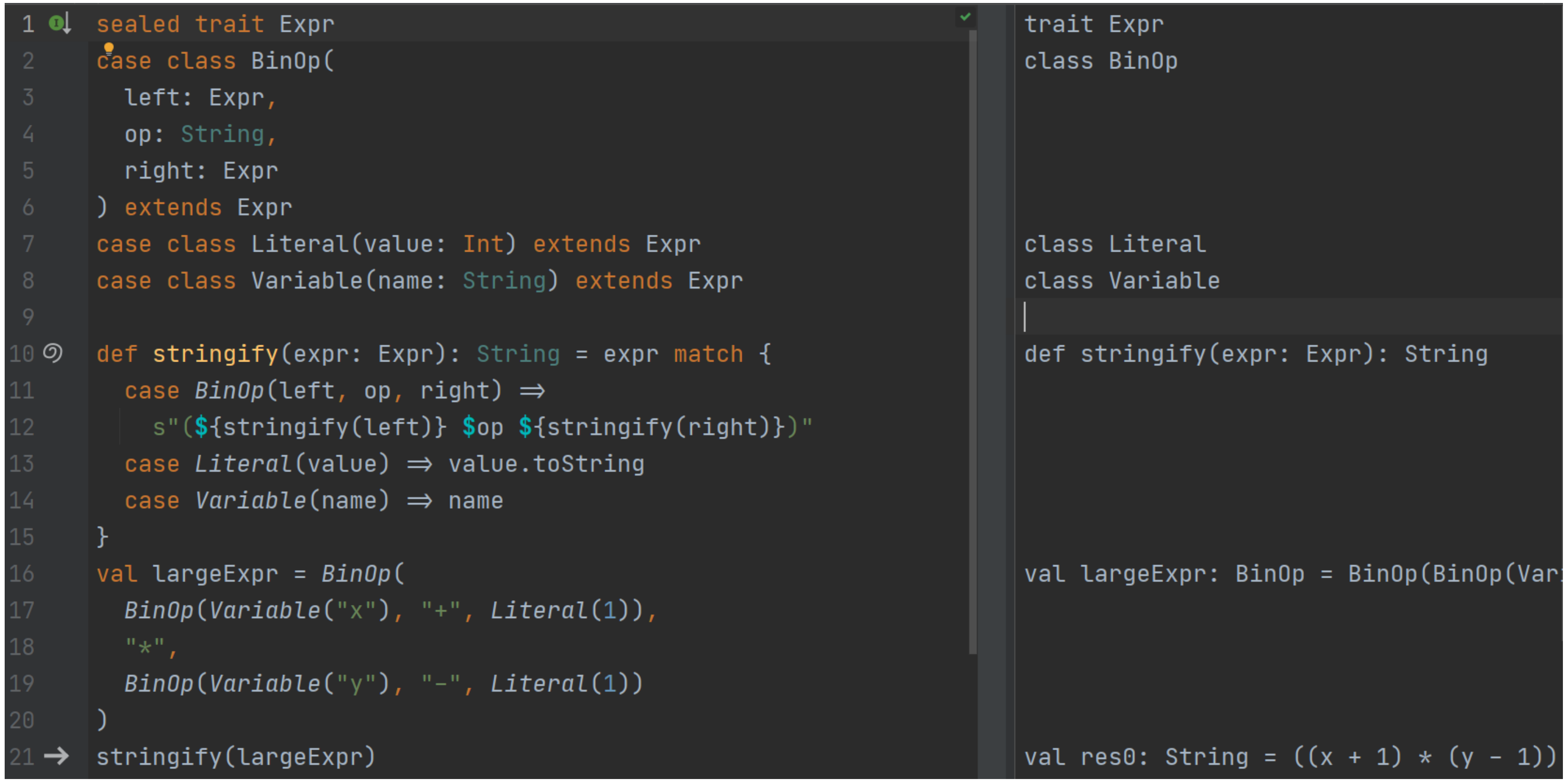
Matches in Loops and Vals



```scala
val a = List((1, "one"), (2, "two"), (3, "three"))
for {
  (i, s) ← a
} println(s + i)

case class Point(x: Int, y: Int)

val p = Point(123, 456)
val Point(x, y) = p

val s"$first $second" = "Hello World"
val flipped = s"$second $first"
val s"$first $second" = "Hello"
```

```
val a: List[(Int, String)] = List((1,one), (2,tw
one1
two2
three3

class Point

val p: Point = Point(123,456)
val x: Int = 123
val y: Int = 456

val first: String = Hello
val second: String = World
val flipped: String = World Hello
scala.MatchError: Hello (of class java.lang.Stri
  ... 39 elided
```

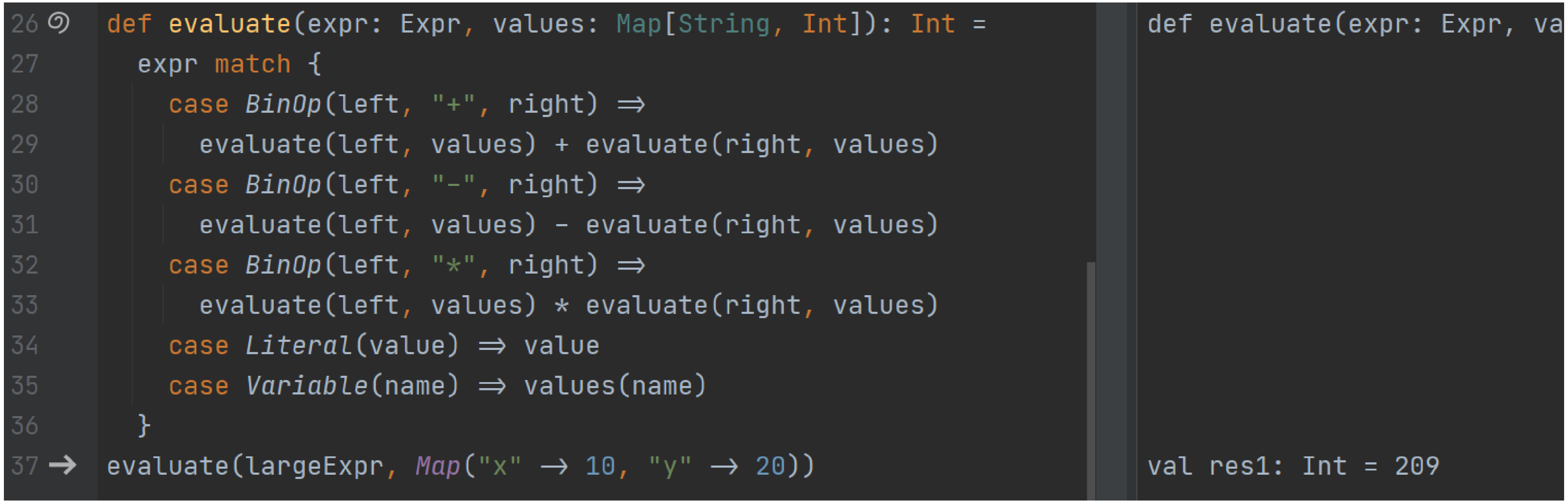
Matching on Sealed Traits



```scala
sealed trait Expr
case class BinOp(
  left: Expr,
  op: String,
  right: Expr
) extends Expr
case class Literal(value: Int) extends Expr
case class Variable(name: String) extends Expr

def stringify(expr: Expr): String = expr match {
  case BinOp(left, op, right) ⇒
    s"(${stringify(left)} $op ${stringify(right)})"
  case Literal(value) ⇒ value.toString
  case Variable(name) ⇒ name
}
val largeExpr = BinOp(
  BinOp(Variable("x"), "+", Literal(1)),
  "*",
  BinOp(Variable("y"), "-", Literal(1))
)
stringify(largeExpr)
```

```
trait Expr
class BinOp
class Literal
class Variable

def stringify(expr: Expr): String
val largeExpr: BinOp = BinOp(BinOp(Var
val res0: String = ((x + 1) * (y - 1))
```

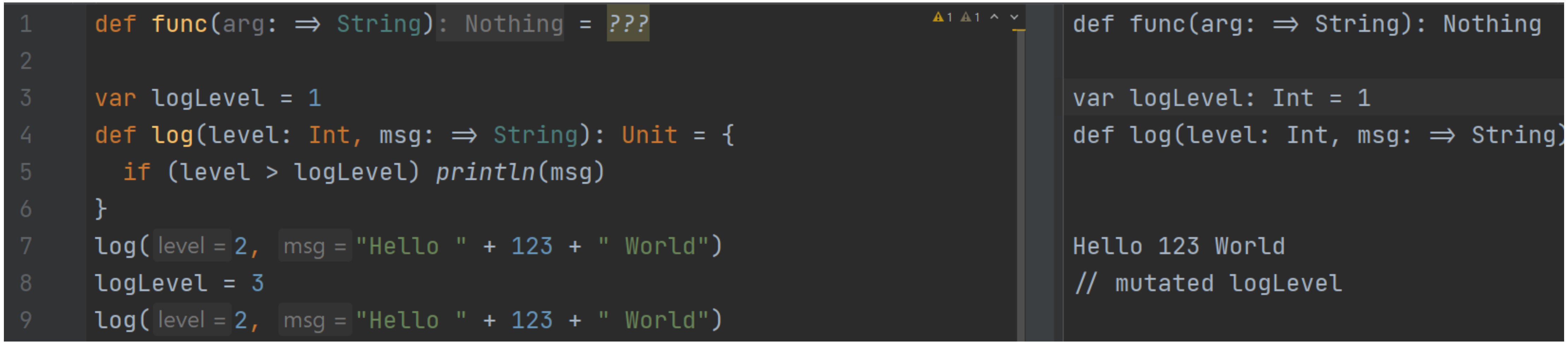
Matching on Sealed Traits 2



```
def evaluate(expr: Expr, values: Map[String, Int]): Int =
  expr match {
    case BinOp(left, "+", right) ⇒
      evaluate(left, values) + evaluate(right, values)
    case BinOp(left, "-", right) ⇒
      evaluate(left, values) - evaluate(right, values)
    case BinOp(left, "*", right) ⇒
      evaluate(left, values) * evaluate(right, values)
    case Literal(value) ⇒ value
    case Variable(name) ⇒ values(name)
  }
evaluate(largeExpr, Map("x" → 10, "y" → 20))
```

```
def evaluate(expr: Expr, va

val res1: Int = 209
```

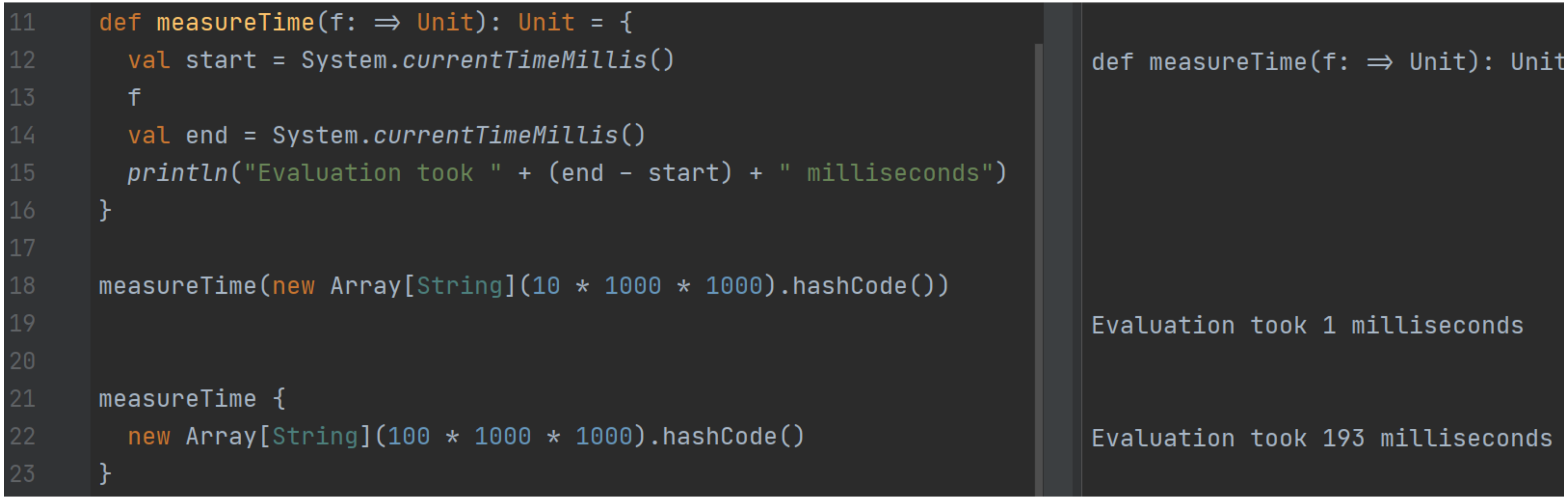
By-Name Parameters



```scala
def func(arg: ⇒ String): Nothing = ???

var logLevel = 1
def log(level: Int, msg: ⇒ String): Unit = {
  if (level > logLevel) println(msg)
}
log(level = 2, msg = "Hello " + 123 + " World")
logLevel = 3
log(level = 2, msg = "Hello " + 123 + " World")
```

```
def func(arg: ⇒ String): Nothing

var logLevel: Int = 1
def log(level: Int, msg: ⇒ String)


Hello 123 World
// mutated logLevel
```

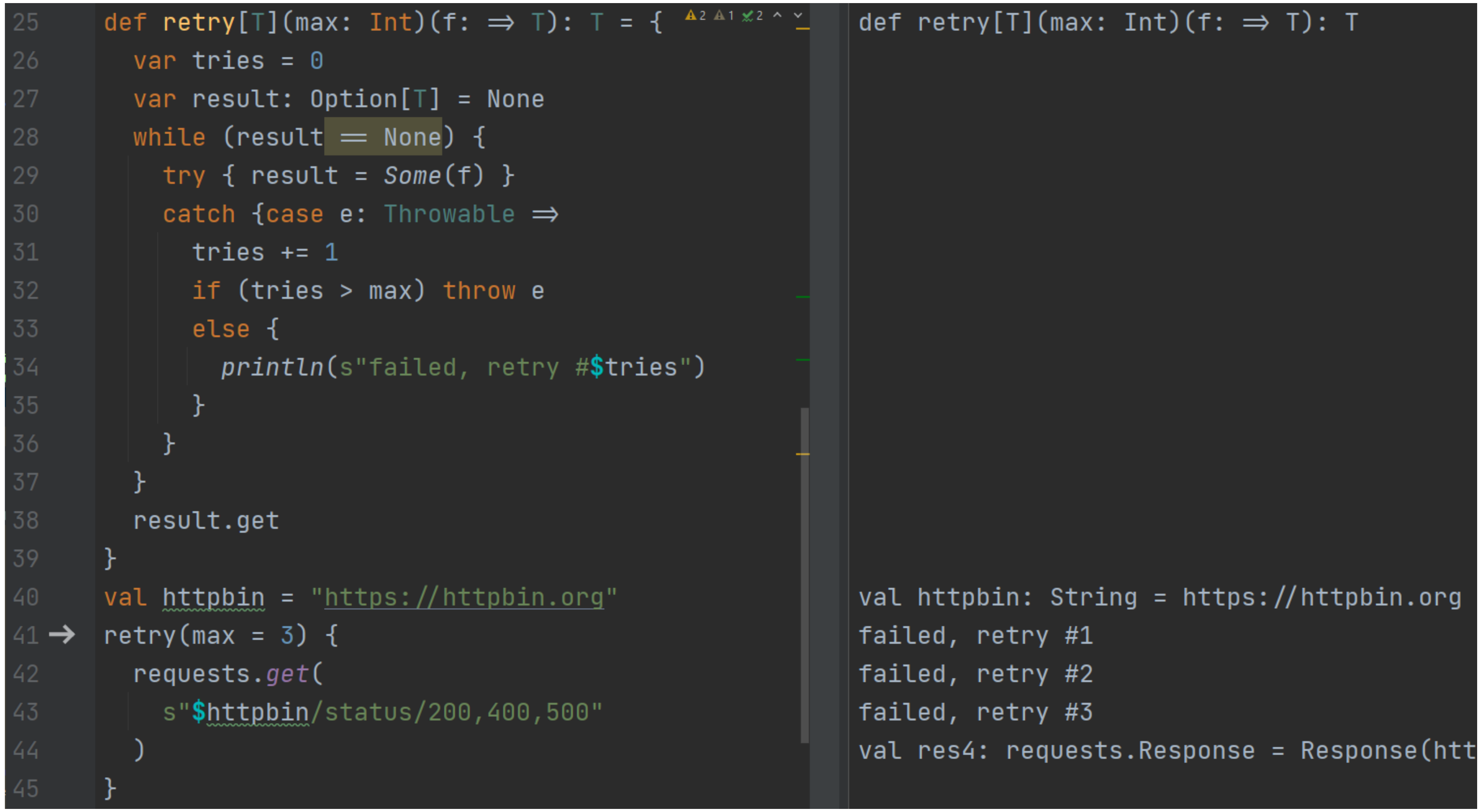
Wrapping Evaluation



```scala
def measureTime(f: ⇒ Unit): Unit = {
  val start = System.currentTimeMillis()
  f
  val end = System.currentTimeMillis()
  println("Evaluation took " + (end - start) + " milliseconds")
}

measureTime(new Array[String](10 * 1000 * 1000).hashCode())


measureTime {
  new Array[String](100 * 1000 * 1000).hashCode()
}
```

```
def measureTime(f: ⇒ Unit): Unit

Evaluation took 1 milliseconds

Evaluation took 193 milliseconds
```

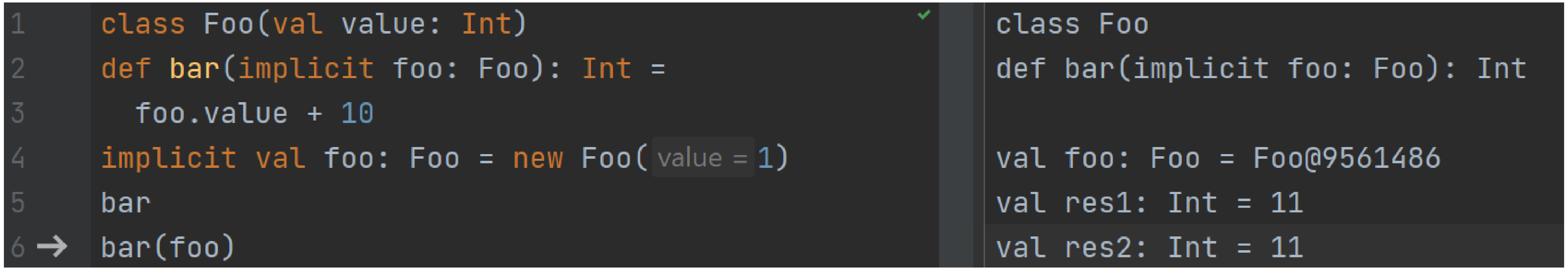
Repeating Evaluation



```scala
def retry[T](max: Int)(f: => T): T = {
  var tries = 0
  var result: Option[T] = None
  while (result == None) {
    try { result = Some(f) }
    catch {case e: Throwable =>
      tries += 1
      if (tries > max) throw e
      else {
        println(s"failed, retry #$tries")
      }
    }
  }
  result.get
}
val httpbin = "https://httpbin.org"
retry(max = 3) {
  requests.get(
    s"$httpbin/status/200,400,500"
  )
}
```

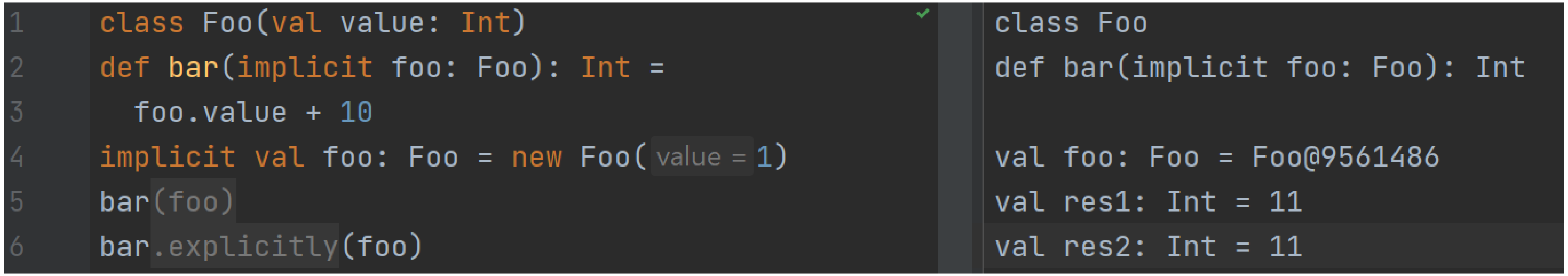
```
def retry[T](max: Int)(f: => T): T









val httpbin: String = https://httpbin.org
failed, retry #1
failed, retry #2
failed, retry #3
val res4: requests.Response = Response(htt
```

Implicit Parameters



```scala
class Foo(val value: Int)
def bar(implicit foo: Foo): Int =
  foo.value + 10
implicit val foo: Foo = new Foo( value = 1)
bar
bar(foo)
```

```
class Foo
def bar(implicit foo: Foo): Int

val foo: Foo = Foo@9561486
val res1: Int = 11
val res2: Int = 11
```



```scala
class Foo(val value: Int)
def bar(implicit foo: Foo): Int =
  foo.value + 10
implicit val foo: Foo = new Foo( value = 1)
bar(foo)
bar.explicitly(foo)
```

```
class Foo
def bar(implicit foo: Foo): Int

val foo: Foo = Foo@9561486
val res1: Int = 11
val res2: Int = 11
```

Useful links

- Mutable collections - http://scalatutorials.com/tour/interactive_tour_of_scala_mutable_collections.html
- Scala implicits - https://riptutorial.com/scala/topic/1732/implicits