Functional Data Structures I

Sequence, Map, Set

Type text here

06016415 Functional Programming

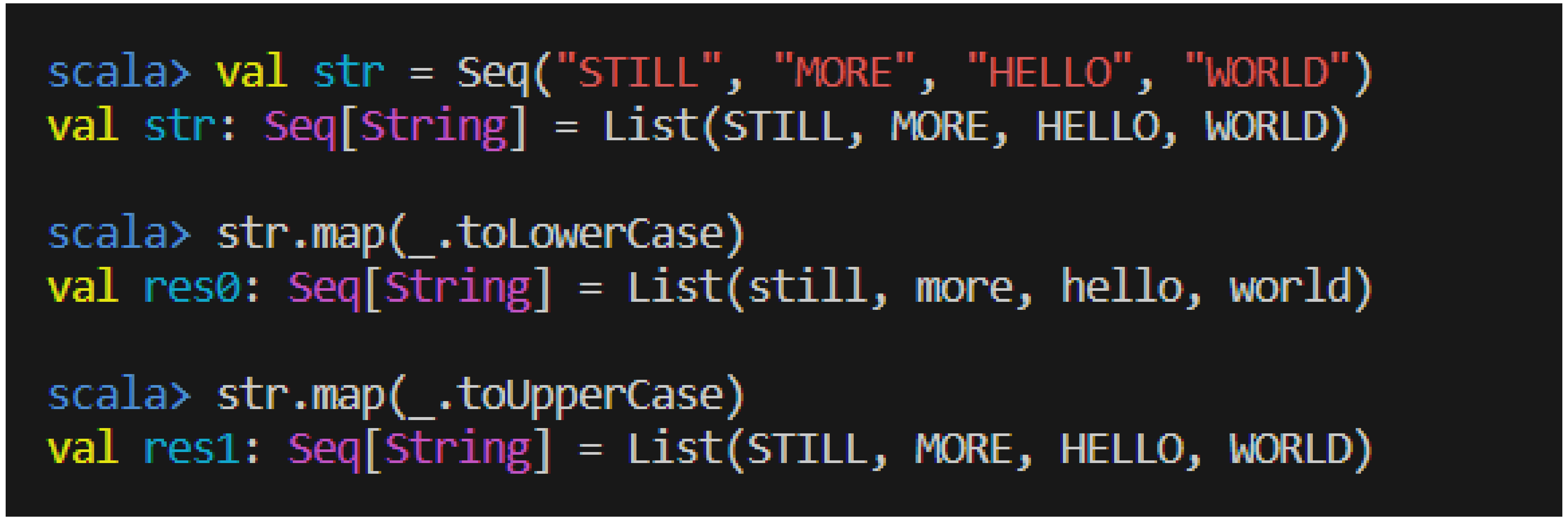
Outline

- Review Scala Syntax
- Sequence, Map, Set

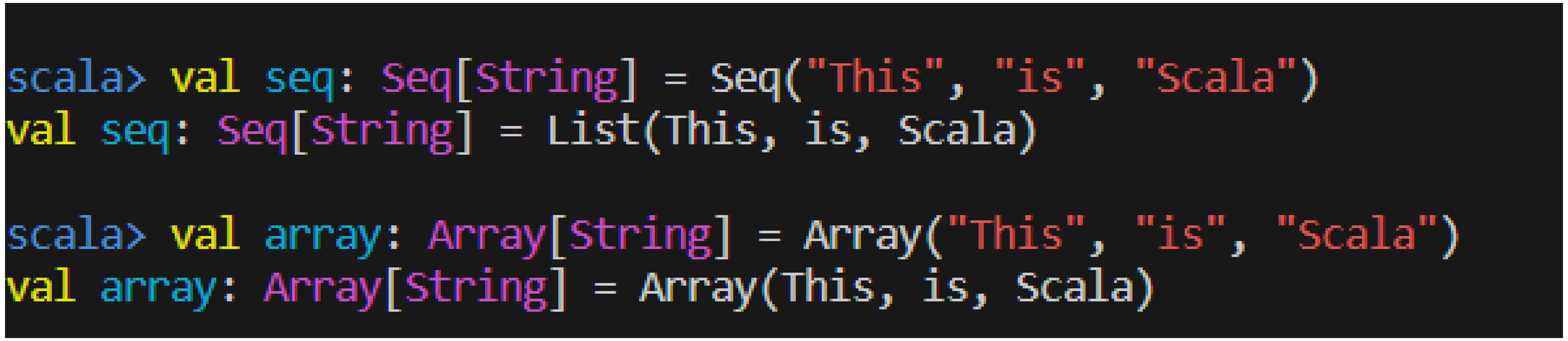
Review Scala Syntax

- **Scala**
  - Hybrid object-functional language
  - Not require functions to be pure
  - Write your code this way whenever possible

Variable Declarations



```
scala> val str = Seq("STILL", "MORE", "HELLO", "WORLD")
val str: Seq[String] = List(STILL, MORE, HELLO, WORLD)

scala> str.map(_.toLowerCase)
val res0: Seq[String] = List(still, more, hello, world)

scala> str.map(_.toUpperCase)
val res1: Seq[String] = List(STILL, MORE, HELLO, WORLD)
```

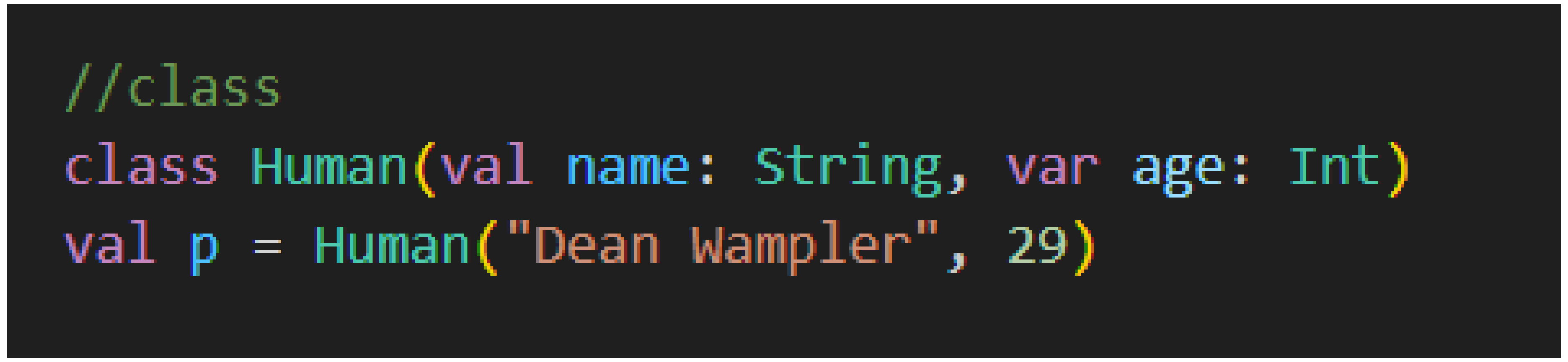


```
scala> val seq: Seq[String] = Seq("This", "is", "Scala")
val seq: Seq[String] = List(This, is, Scala)

scala> val array: Array[String] = Array("This", "is", "Scala")
val array: Array[String] = Array(This, is, Scala)
```

sequence เป็น immutable = เปลี่ยนแปลงค่าไม่ได้
array เปลี่ยนแปลงค่าได้

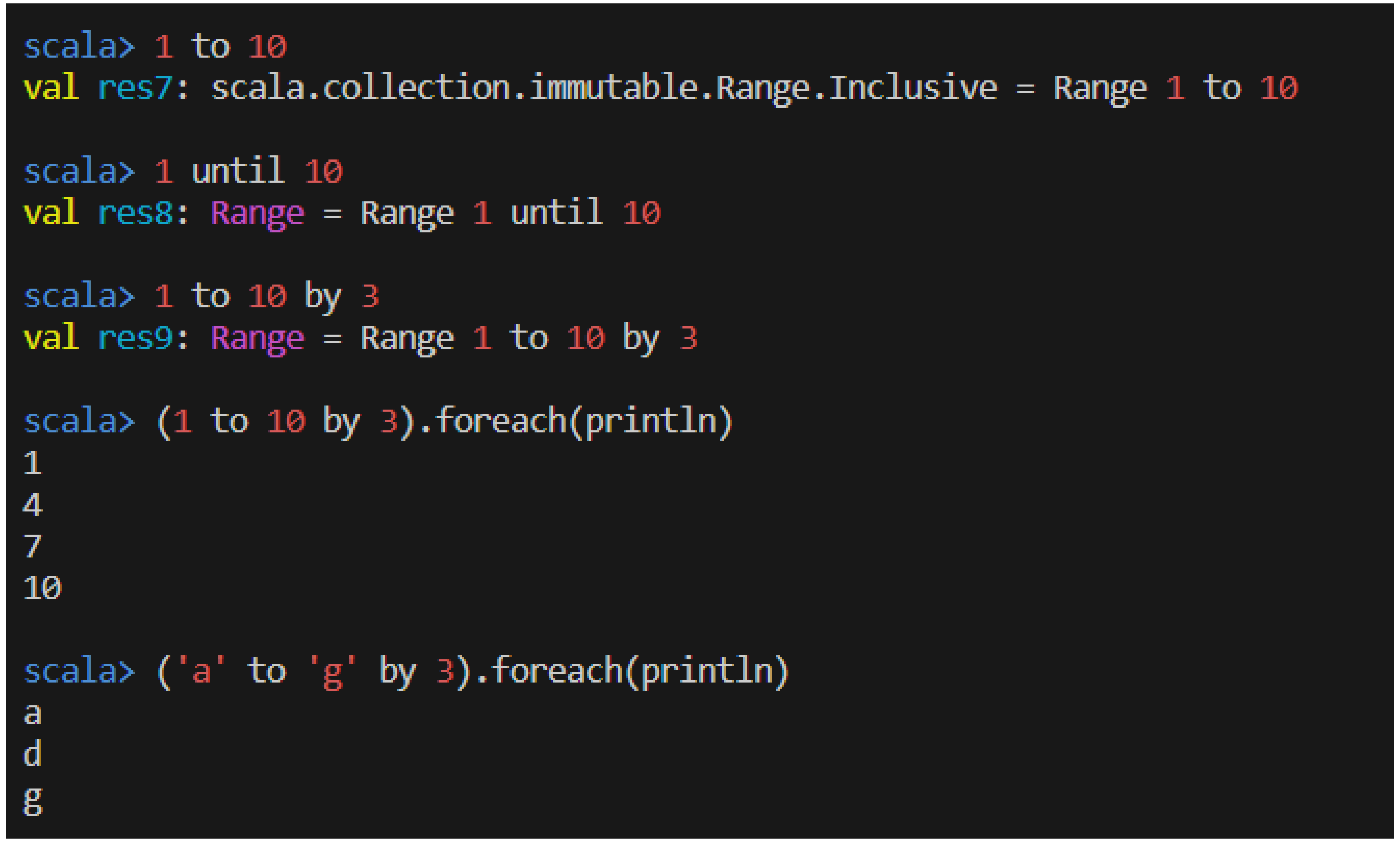
Human class with an immutable name, but a mutable age

mutable เปลี่ยนแปลงค่าได้



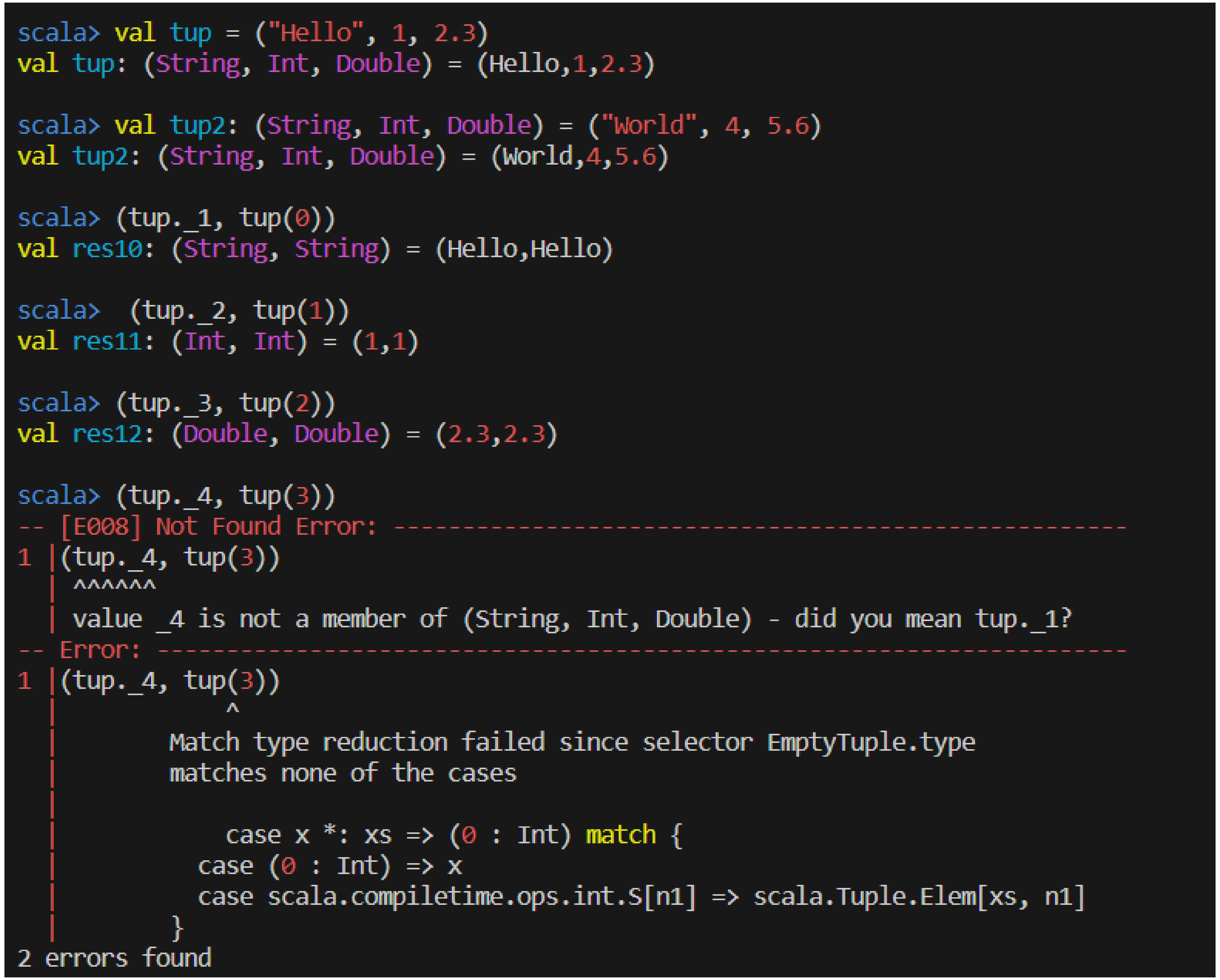
```scala
//class
class Human(val name: String, var age: Int)
val p = Human("Dean Wampler", 29)
```

Ranges



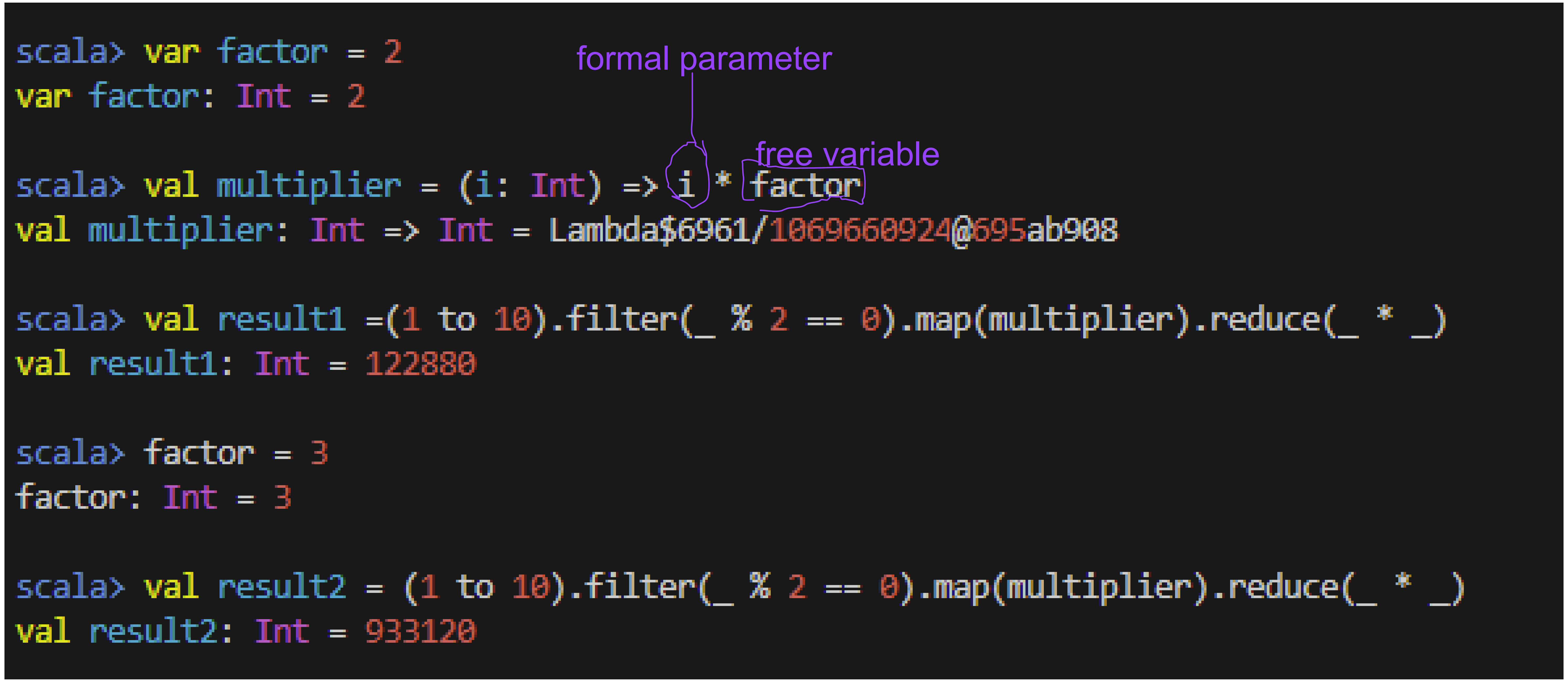
```
scala> 1 to 10
val res7: scala.collection.immutable.Range.Inclusive = Range 1 to 10

scala> 1 until 10
val res8: Range = Range 1 until 10

scala> 1 to 10 by 3
val res9: Range = Range 1 to 10 by 3

scala> (1 to 10 by 3).foreach(println)
1
4
7
10

scala> ('a' to 'g' by 3).foreach(println)
a
d
g
```

Tuples



```
scala> val tup = ("Hello", 1, 2.3)
val tup: (String, Int, Double) = (Hello,1,2.3)

scala> val tup2: (String, Int, Double) = ("World", 4, 5.6)
val tup2: (String, Int, Double) = (World,4,5.6)

scala> (tup._1, tup(0))
val res10: (String, String) = (Hello,Hello)

scala>  (tup._2, tup(1))
val res11: (Int, Int) = (1,1)

scala> (tup._3, tup(2))
val res12: (Double, Double) = (2.3,2.3)

scala> (tup._4, tup(3))
-- [E008] Not Found Error: -----------------------------------------------------
1 |(tup._4, tup(3))
  | ^^^^^^
  | value _4 is not a member of (String, Int, Double) - did you mean tup._1?
-- Error: ----------------------------------------------------------------------
1 |(tup._4, tup(3))
  |            ^
  |        Match type reduction failed since selector EmptyTuple.type
  |        matches none of the cases
  |
  |            case x *: xs => (0 : Int) match {
  |          case (0 : Int) => x
  |          case scala.compiletime.ops.int.S[n1] => scala.Tuple.Elem[xs, n1]
  |        }
2 errors found
```

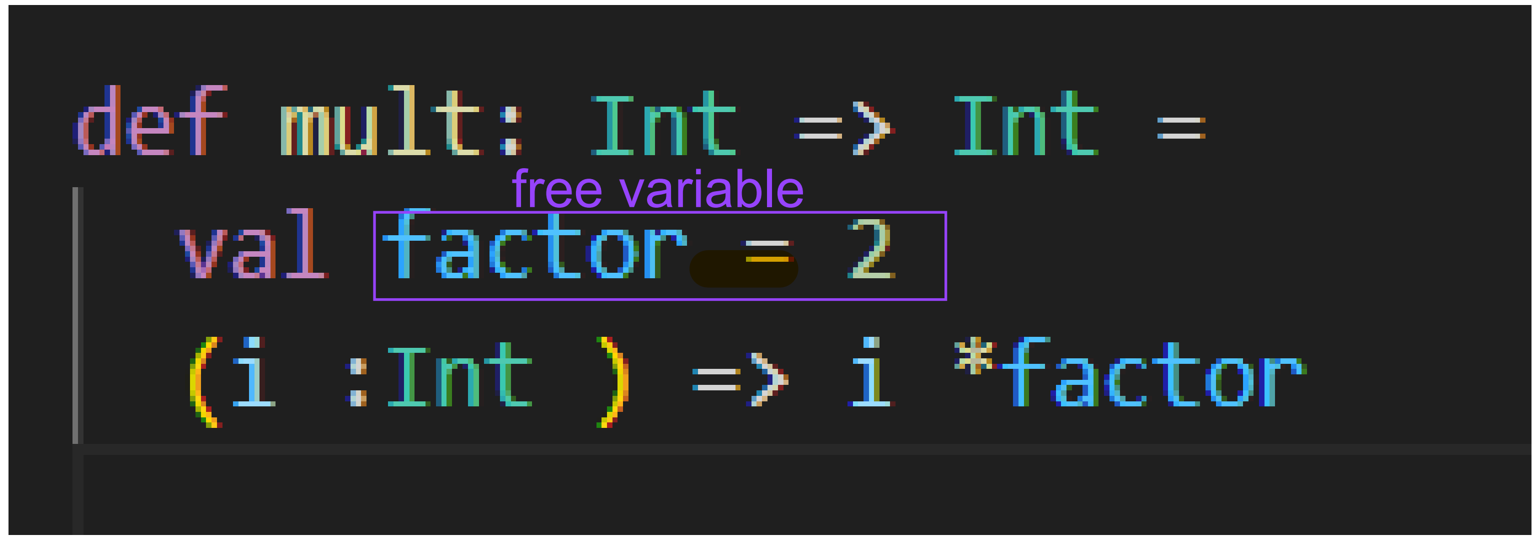
Anonymous Functions



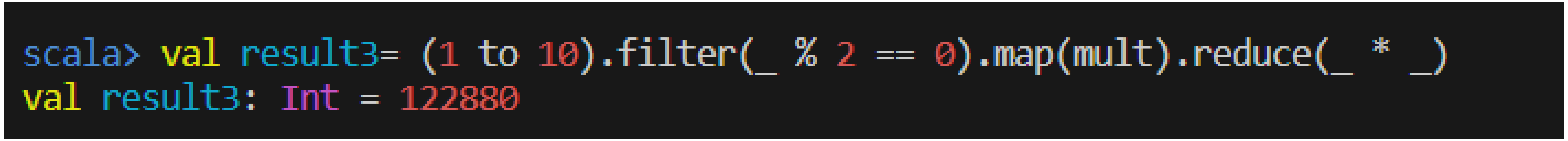
```
scala> var factor = 2
var factor: Int = 2

scala> val multiplier = (i: Int) => i * factor
val multiplier: Int => Int = Lambda$6961/1069660924@695ab908

scala> val result1 =(1 to 10).filter(_ % 2 == 0).map(multiplier).reduce(_ * _)
val result1: Int = 122880

scala> factor = 3
factor: Int = 3

scala> val result2 = (1 to 10).filter(_ % 2 == 0).map(multiplier).reduce(_ * _)
val result2: Int = 933120
```

Anonymous Functions

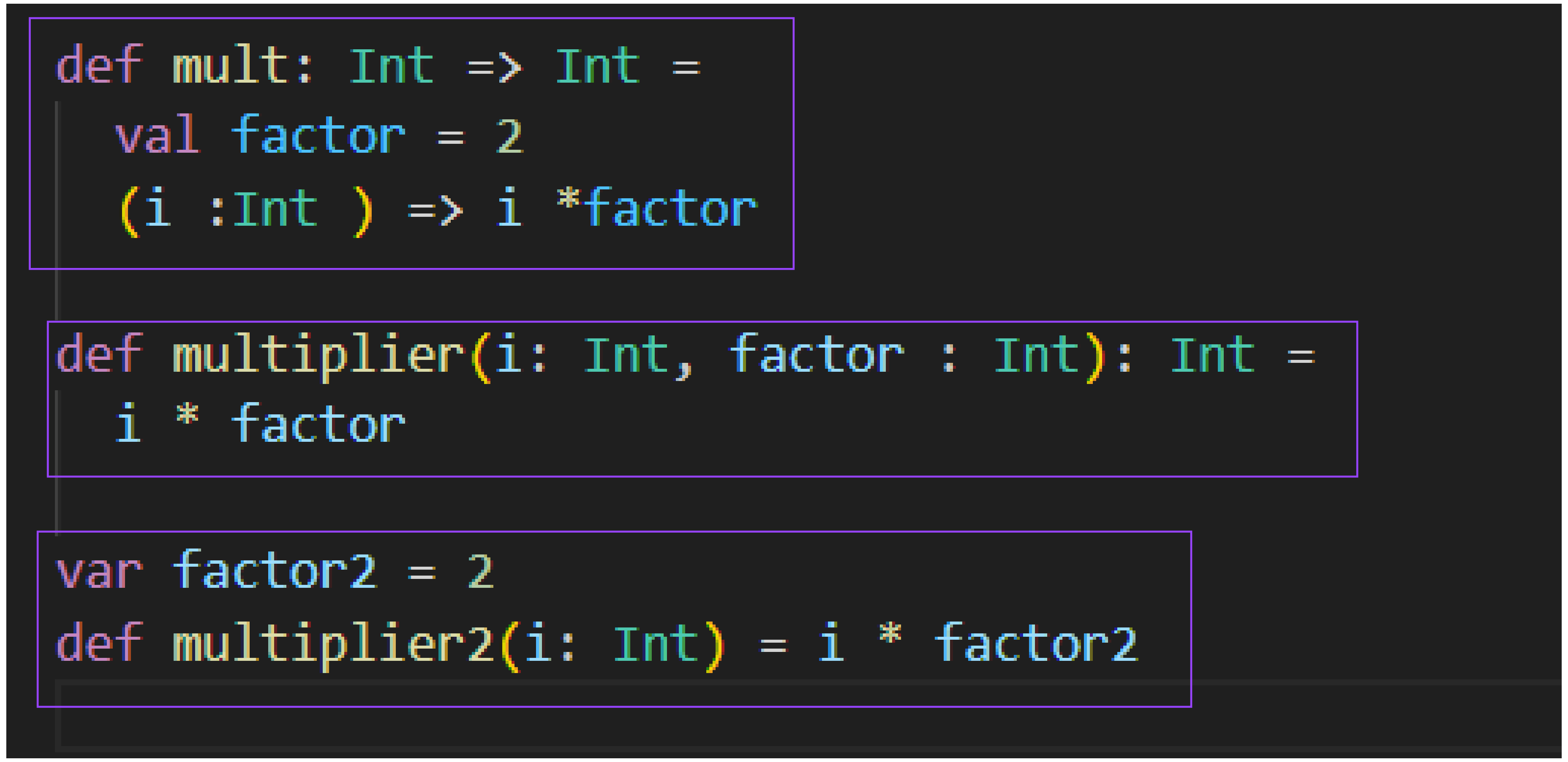
ไม่เปี๊ๆ functional programming

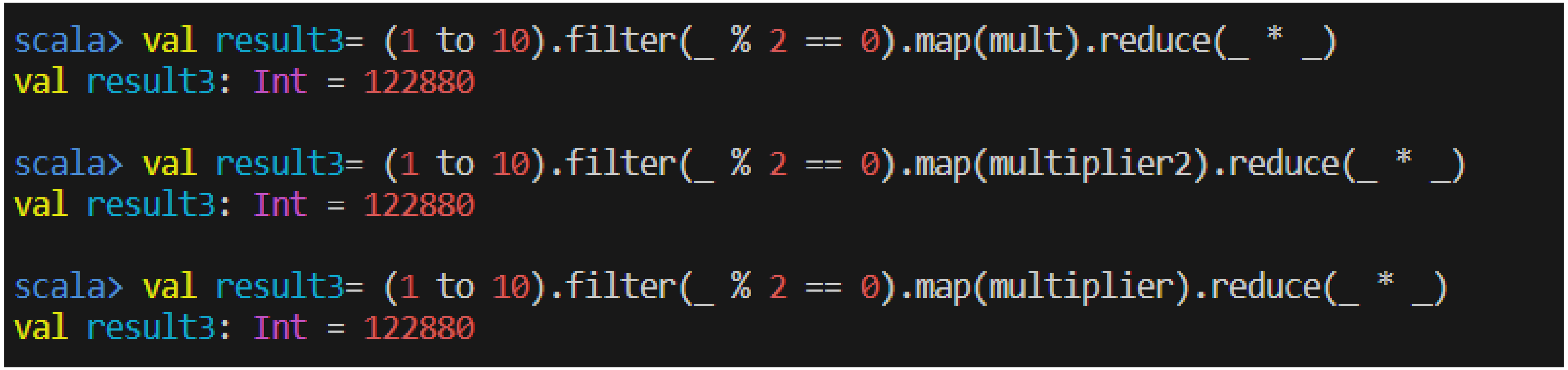


```
def mult: Int => Int =
  val factor = 2
  (i :Int ) => i *factor
```



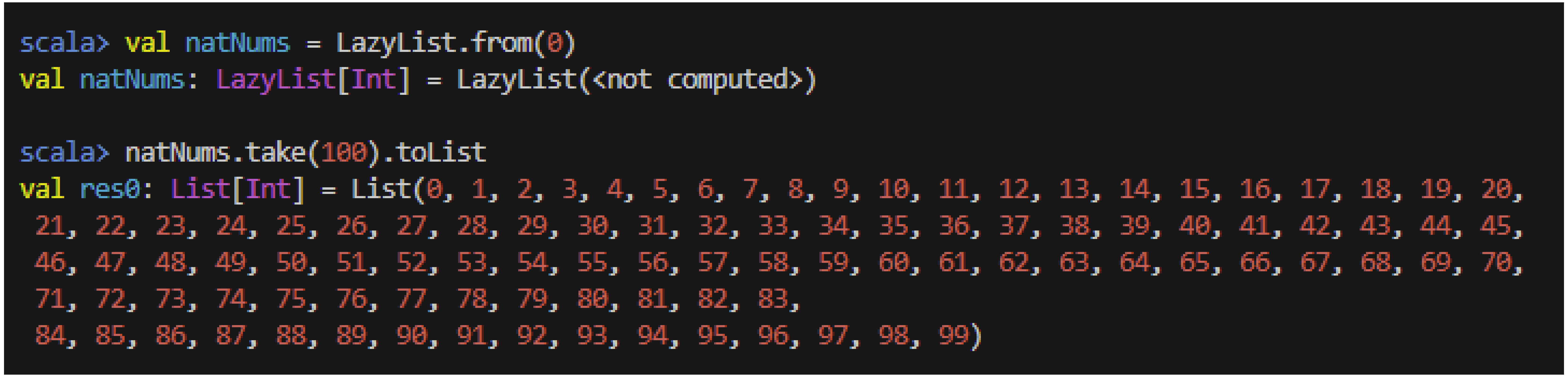
```
scala> val result3= (1 to 10).filter(_ % 2 == 0).map(mult).reduce(_ * _)
val result3: Int = 122880
```


1
```
def mult: Int => Int =
  val factor = 2
  (i :Int ) => i *factor
```
2
```
def multiplier(i: Int, factor : Int): Int =
  i * factor
```
3
```
var factor2 = 2
def multiplier2(i: Int) = i * factor2
```


```
scala> val result3= (1 to 10).filter(_ % 2 == 0).map(mult).reduce(_ * _)
val result3: Int = 122880

scala> val result3= (1 to 10).filter(_ % 2 == 0).map(multiplier2).reduce(_ * _)
val result3: Int = 122880

scala> val result3= (1 to 10).filter(_ % 2 == 0).map(multiplier).reduce(_ * _)
val result3: Int = 122880
```

LazyList in the standard library

Like many of the other data structures we’ve seen so far,
LazyList exists in the Scala standard library (see the API at https://mng.bz/M00D).



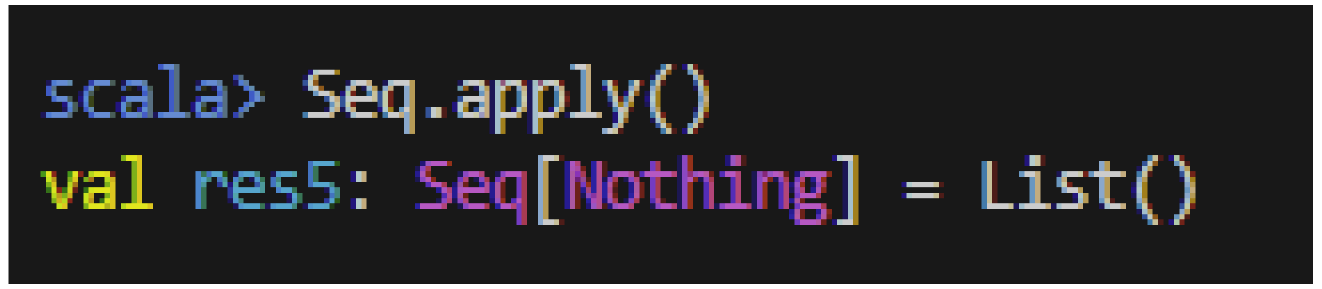
```
scala> val natNums = LazyList.from(0)
val natNums: LazyList[Int] = LazyList(<not computed>)

scala> natNums.take(100).toList
val res0: List[Int] = List(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83,
 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99)
```

Functional Data Structures

- Functional programming จะเน้นการใช้ ความรู้พื้นฐานด้าน Data Structures and Algorithms
- Add, Changes, Remove โดยไม่แก้ไข โครงสร้างข้อมูลเดิมในโปรแกรม

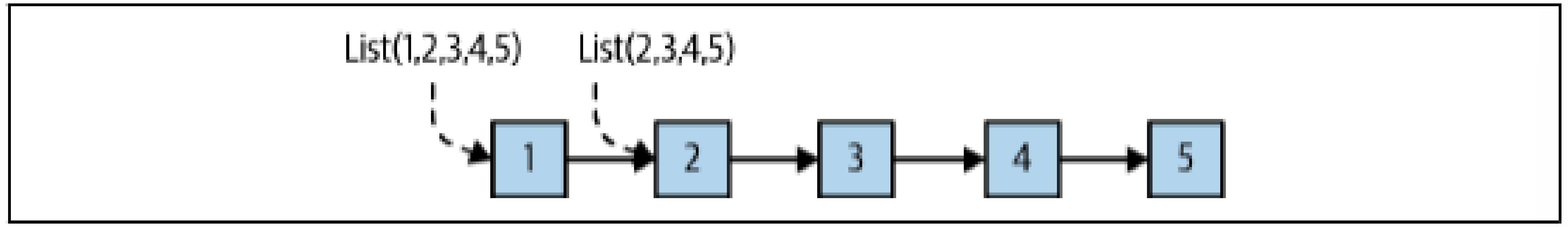
Sequences

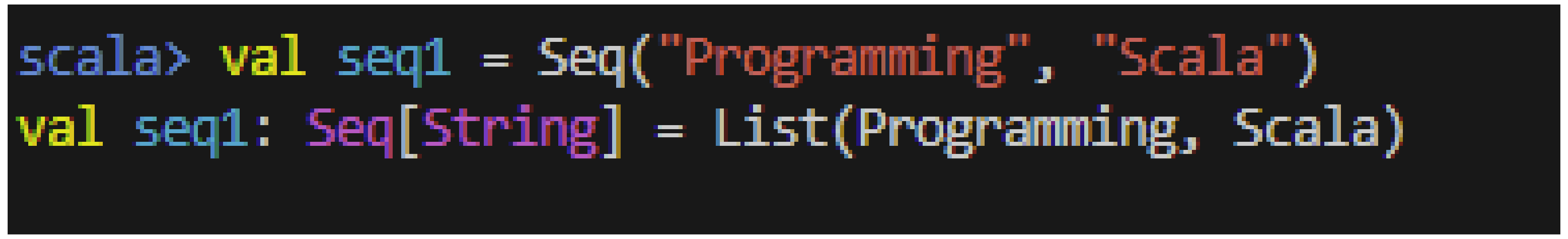
- Data Structures โดยทั่วไปมักกำหนดให้มีโครงสร้างแบบ Sequential (ตามลำดับ)
- Array, List, Seq, Vector



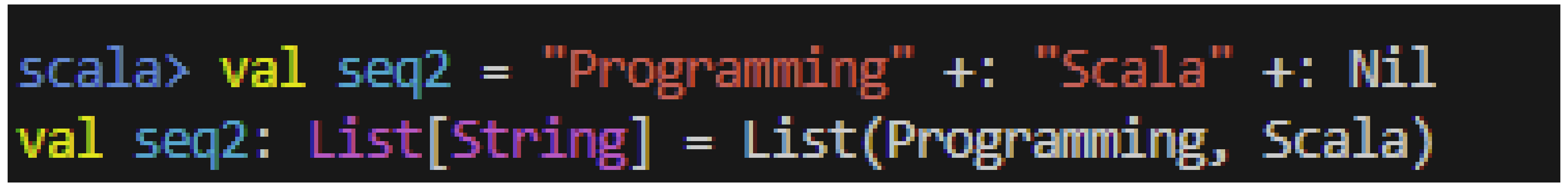
```
scala> Seq.apply()
val res5: Seq[Nothing] = List()
```

Lists are immutable



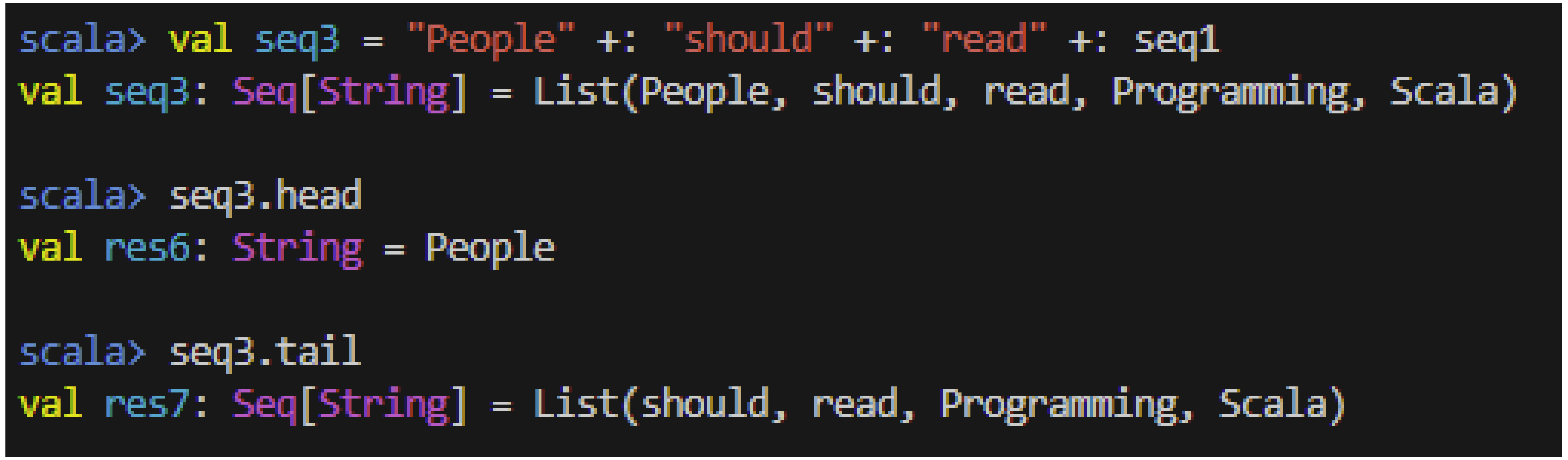


```
scala> val seq1 = Seq("Programming", "Scala")
val seq1: Seq[String] = List(Programming, Scala)
```



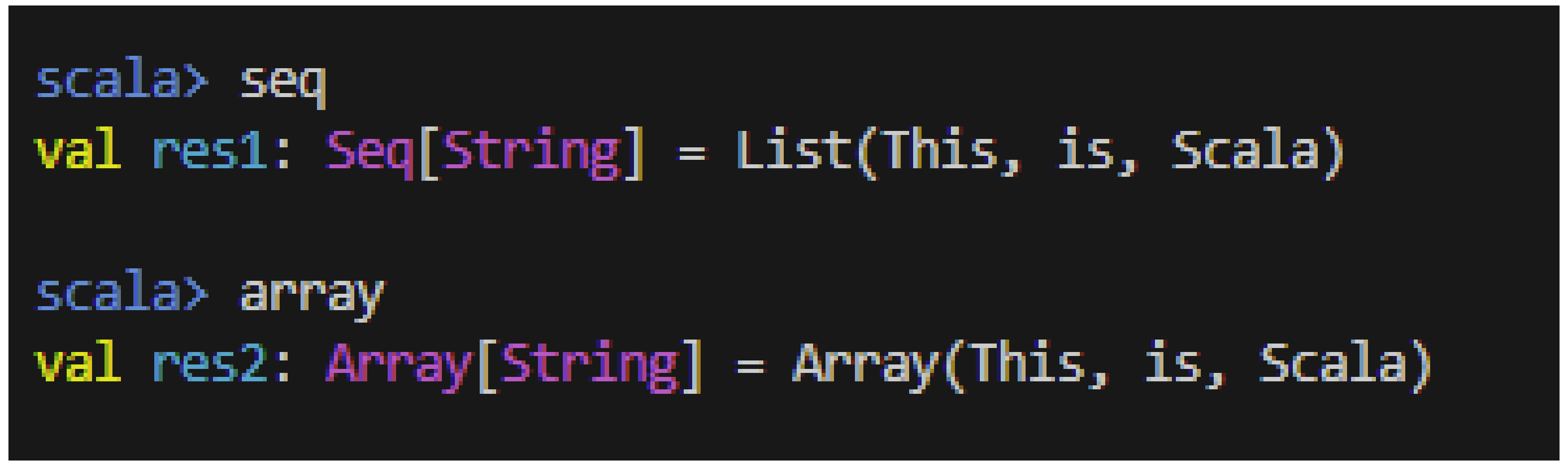
```
scala> val seq2 = "Programming" +: "Scala" +: Nil
val seq2: List[String] = List(Programming, Scala)
```

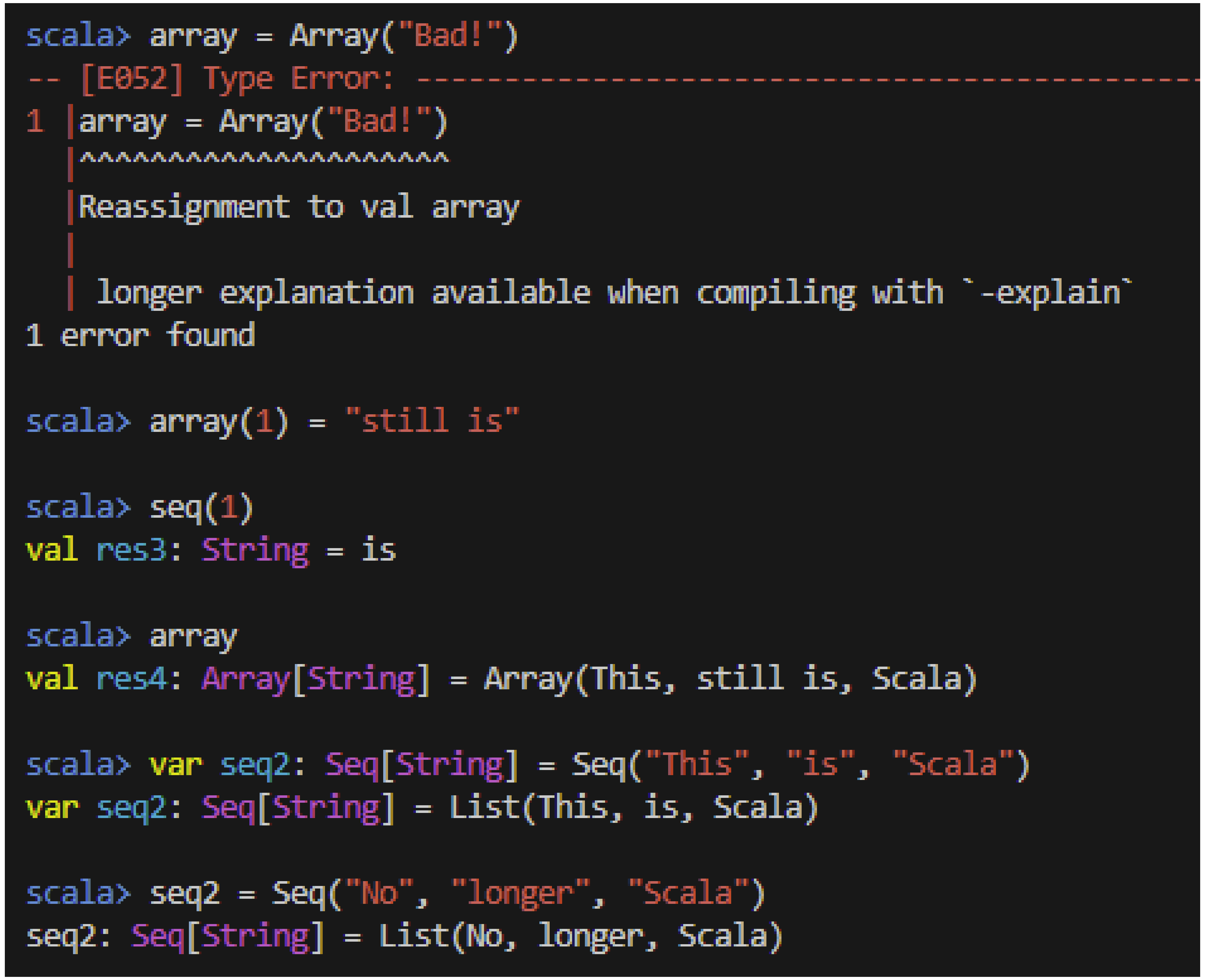
Nil is a subtype of List that is a convenient object when an empty list is required.



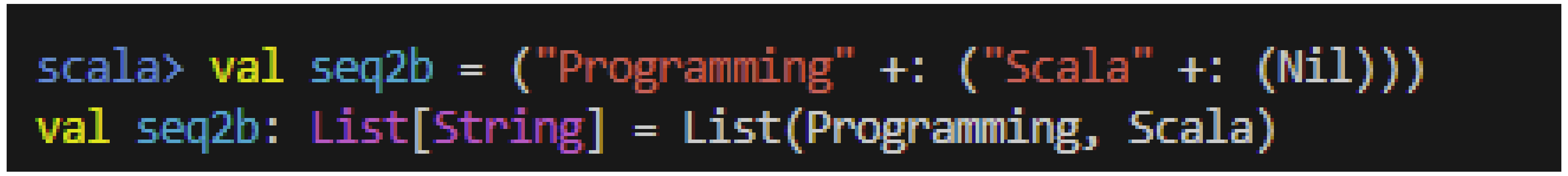
```
scala> val seq3 = "People" +: "should" +: "read" +: seq1
val seq3: Seq[String] = List(People, should, read, Programming, Scala)

scala> seq3.head
val res6: String = People

scala> seq3.tail
val res7: Seq[String] = List(should, read, Programming, Scala)
```

+: (prepend), and :+ (append)


```
scala> seq
val res1: Seq[String] = List(This, is, Scala)

scala> array
val res2: Array[String] = Array(This, is, Scala)
```

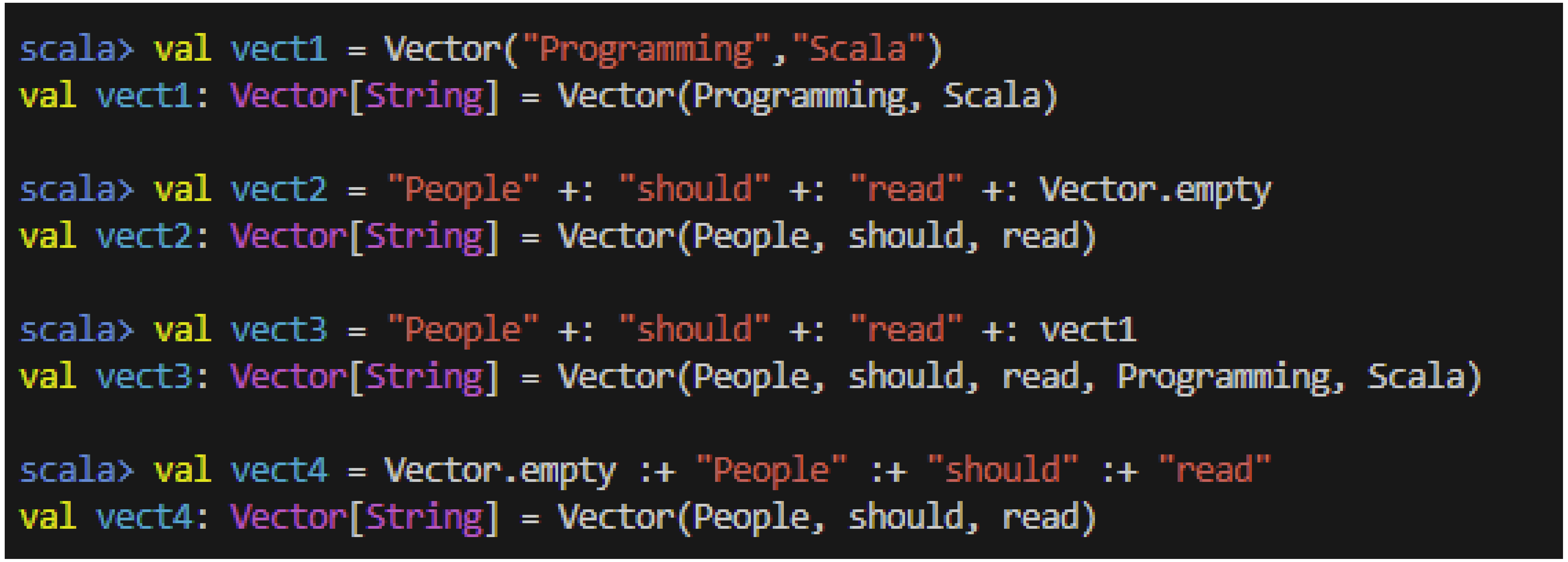

```
scala> array = Array("Bad!")
-- [E052] Type Error: ----------------------------------------------
1 |array = Array("Bad!")
  |^^^^^^^^^^^^^^^^^^^^^
  |Reassignment to val array
  |
  | longer explanation available when compiling with `-explain`
1 error found

scala> array(1) = "still is"

scala> seq(1)
val res3: String = is

scala> array
val res4: Array[String] = Array(This, still is, Scala)

scala> var seq2: Seq[String] = Seq("This", "is", "Scala")
var seq2: Seq[String] = List(This, is, Scala)

scala> seq2 = Seq("No", "longer", "Scala")
seq2: Seq[String] = List(No, longer, Scala)
```

เพิ่มเครื่องหมาย () เพื่อจัดลำดับในการสร้างตัวแปรได้



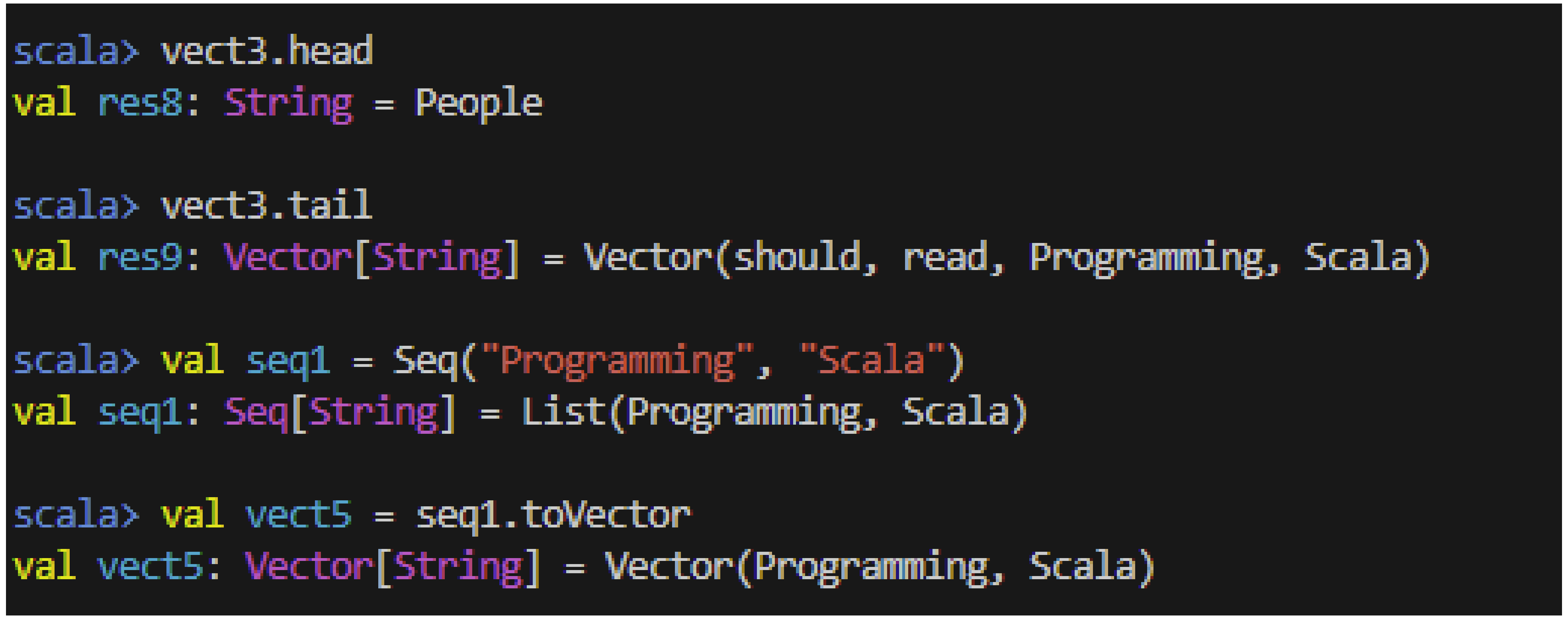
```
scala> val seq2b = ("Programming" +: ("Scala" +: (Nil)))
val seq2b: List[String] = List(Programming, Scala)
```

Vector คุณสมบัติเป็นโครงสร้างแบบ dynamic มีลักษณะการจัดเก็บข้อมูลคล้ายกับ Array แต่ vector มีความหยืดหยุ่น และมีประสิทธิภาพมากกว่า



```
scala> val vect1 = Vector("Programming","Scala")
val vect1: Vector[String] = Vector(Programming, Scala)

scala> val vect2 = "People" +: "should" +: "read" +: Vector.empty
val vect2: Vector[String] = Vector(People, should, read)

scala> val vect3 = "People" +: "should" +: "read" +: vect1
val vect3: Vector[String] = Vector(People, should, read, Programming, Scala)

scala> val vect4 = Vector.empty :+ "People" :+ "should" :+ "read"
val vect4: Vector[String] = Vector(People, should, read)
```

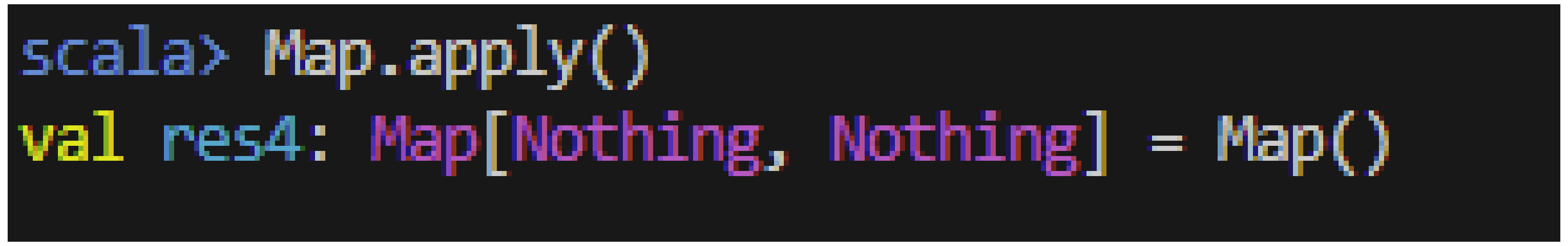
+: (prepend), and :+ (append)



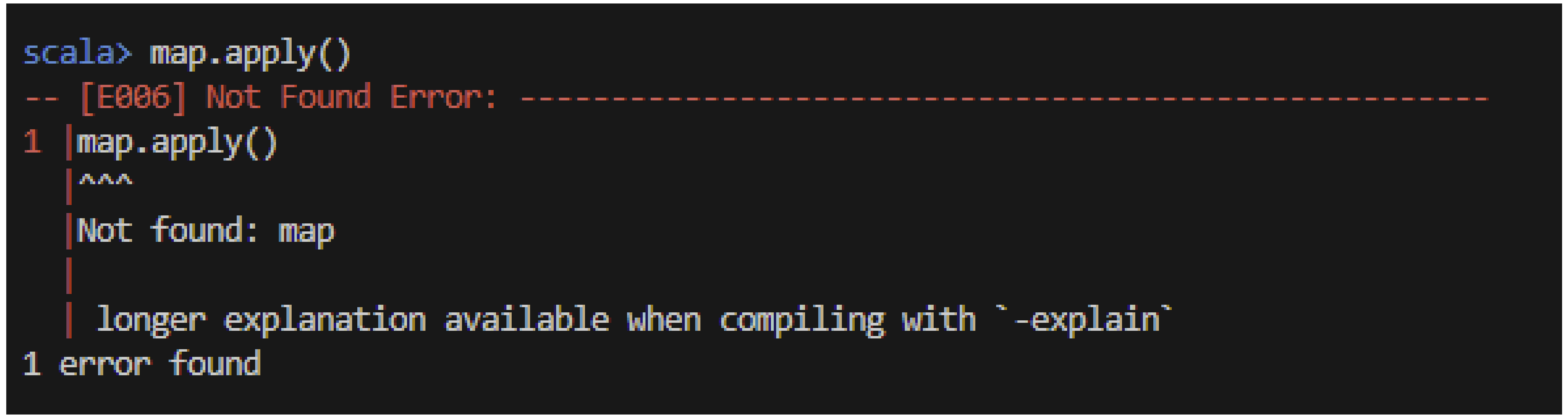
```
scala> vect3.head
val res8: String = People

scala> vect3.tail
val res9: Vector[String] = Vector(should, read, Programming, Scala)

scala> val seq1 = Seq("Programming", "Scala")
val seq1: Seq[String] = List(Programming, Scala)

scala> val vect5 = seq1.toVector
val vect5: Vector[String] = Vector(Programming, Scala)
```

Maps

- Maps เป็น โครงสร้างข้อมูลพื้นฐานทั่วไป เพื่อใช้จับคู่ระหว่าง Key กับ Value
- Key ต้องเป็น unique

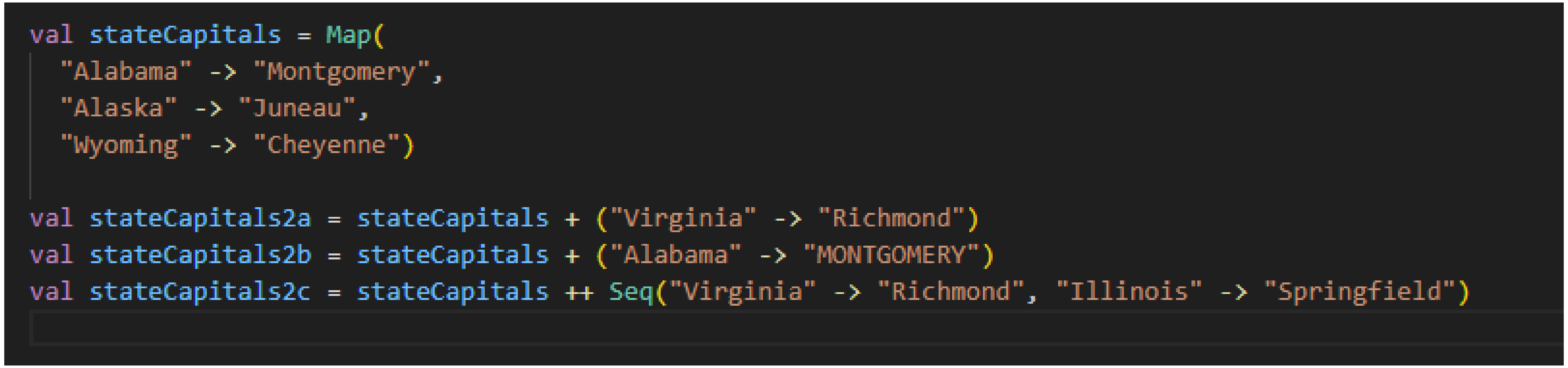


```
scala> Map.apply()
val res4: Map[Nothing, Nothing] = Map()
```

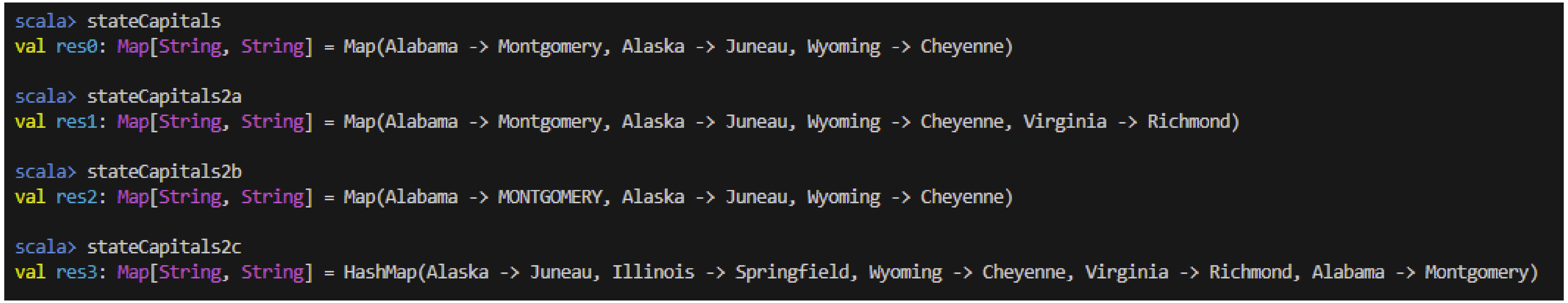


```
scala> map.apply()
-- [E006] Not Found Error: -----------------------------------------------------
1 |map.apply()
  |^^^
  |Not found: map
  |
  | longer explanation available when compiling with `-explain`
1 error found
```

- คล้ายกับ Seq แต่มีลักษณะการใช้งานต่างกัน



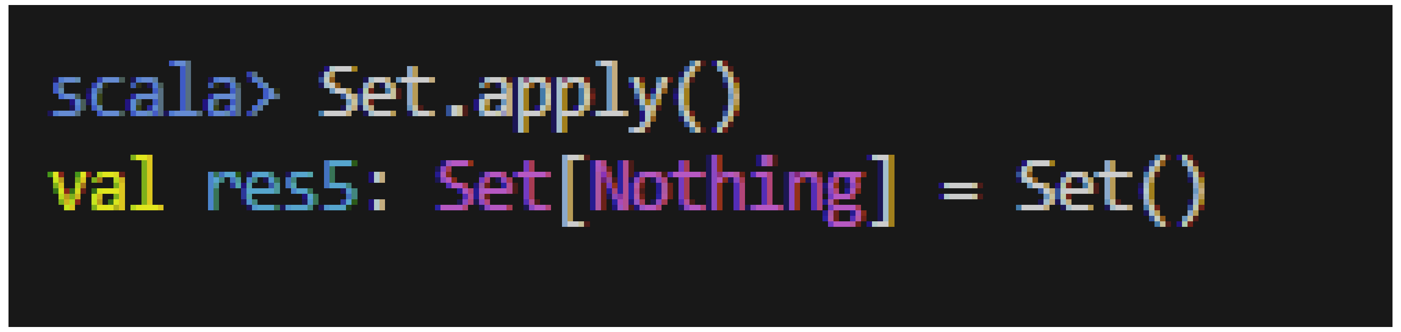
```scala
val stateCapitals = Map(
  "Alabama" -> "Montgomery",
  "Alaska" -> "Juneau",
  "Wyoming" -> "Cheyenne")

val stateCapitals2a = stateCapitals + ("Virginia" -> "Richmond")
val stateCapitals2b = stateCapitals + ("Alabama" -> "MONTGOMERY")
val stateCapitals2c = stateCapitals ++ Seq("Virginia" -> "Richmond", "Illinois" -> "Springfield")
```



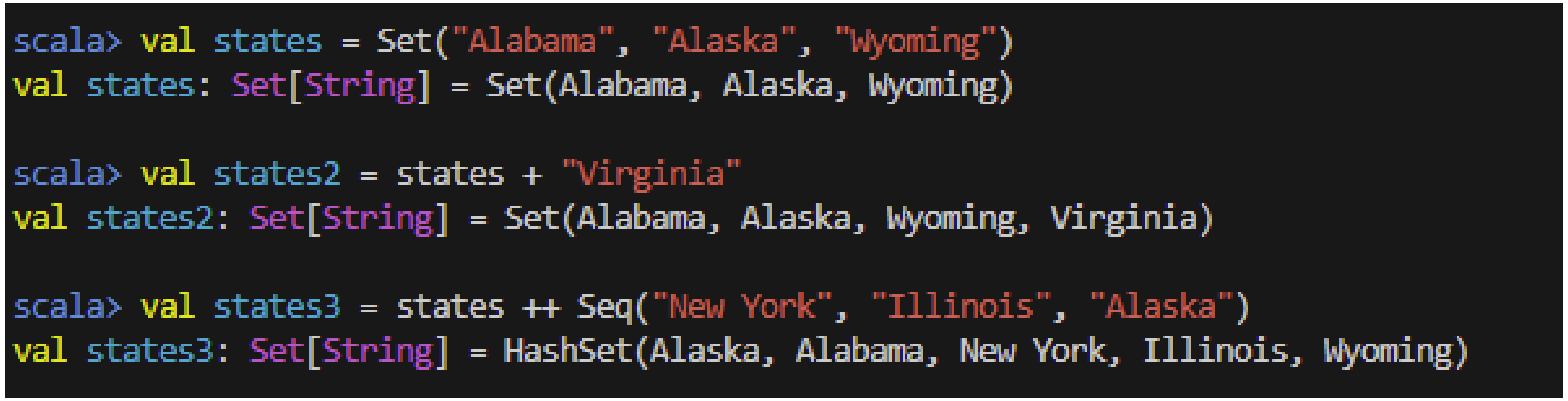
```
scala> stateCapitals
val res0: Map[String, String] = Map(Alabama -> Montgomery, Alaska -> Juneau, Wyoming -> Cheyenne)

scala> stateCapitals2a
val res1: Map[String, String] = Map(Alabama -> Montgomery, Alaska -> Juneau, Wyoming -> Cheyenne, Virginia -> Richmond)

scala> stateCapitals2b
val res2: Map[String, String] = Map(Alabama -> MONTGOMERY, Alaska -> Juneau, Wyoming -> Cheyenne)

scala> stateCapitals2c
val res3: Map[String, String] = HashMap(Alaska -> Juneau, Illinois -> Springfield, Wyoming -> Cheyenne, Virginia -> Richmond, Alabama -> Montgomery)
```

Sets

- Sets คือ โครงสร้างข้อมูลที่ไม่จัดลำดับ
- Sets คล้ายกับ Maps Key คือ แต่ละ elements ต้องเป็น unique



```
scala> Set.apply()
val res5: Set[Nothing] = Set()
```



```
scala> val states = Set("Alabama", "Alaska", "Wyoming")
val states: Set[String] = Set(Alabama, Alaska, Wyoming)

scala> val states2 = states + "Virginia"
val states2: Set[String] = Set(Alabama, Alaska, Wyoming, Virginia)

scala> val states3 = states ++ Seq("New York", "Illinois", "Alaska")
val states3: Set[String] = HashSet(Alaska, Alabama, New York, Illinois, Wyoming)
```