Pattern Matching

Lesson Objectives

- After completing this lesson, you should be able to:
  - Describe how to use pattern matching to handle different values in different ways
  - Outline how case classes and ADTs help in pattern matching
  - Illustrate how to extract values from tuples

Typesafe

What is Pattern Matching?

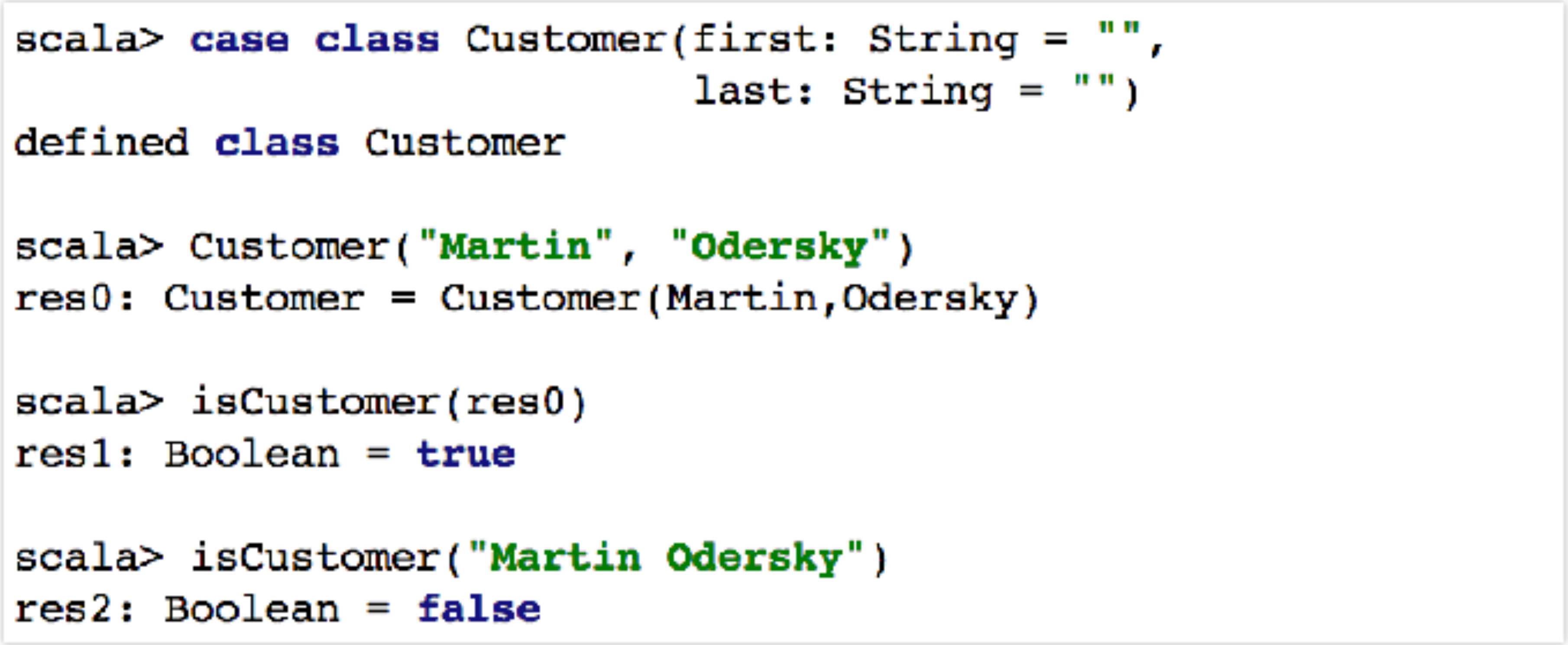
- Many languages have the concept of **`switch`**/**`case`**
- Pattern matching is similar, but can be applied across many different types of data
- Can be embedded within other expressions as a way of cleanly expressing conditional logic

Typesafe

The `match` Keyword

```scala
def isCustomer(someValue: Any): Boolean = {
  someValue match {
    case cust: Customer => true
    case _ => false
  }
}
```

Usage



```
scala> case class Customer(first: String = "",
                           last: String = "")
defined class Customer

scala> Customer("Martin", "Odersky")
res0: Customer = Customer(Martin,Odersky)

scala> isCustomer(res0)
res1: Boolean = true

scala> isCustomer("Martin Odersky")
res2: Boolean = false
```

Typesafe

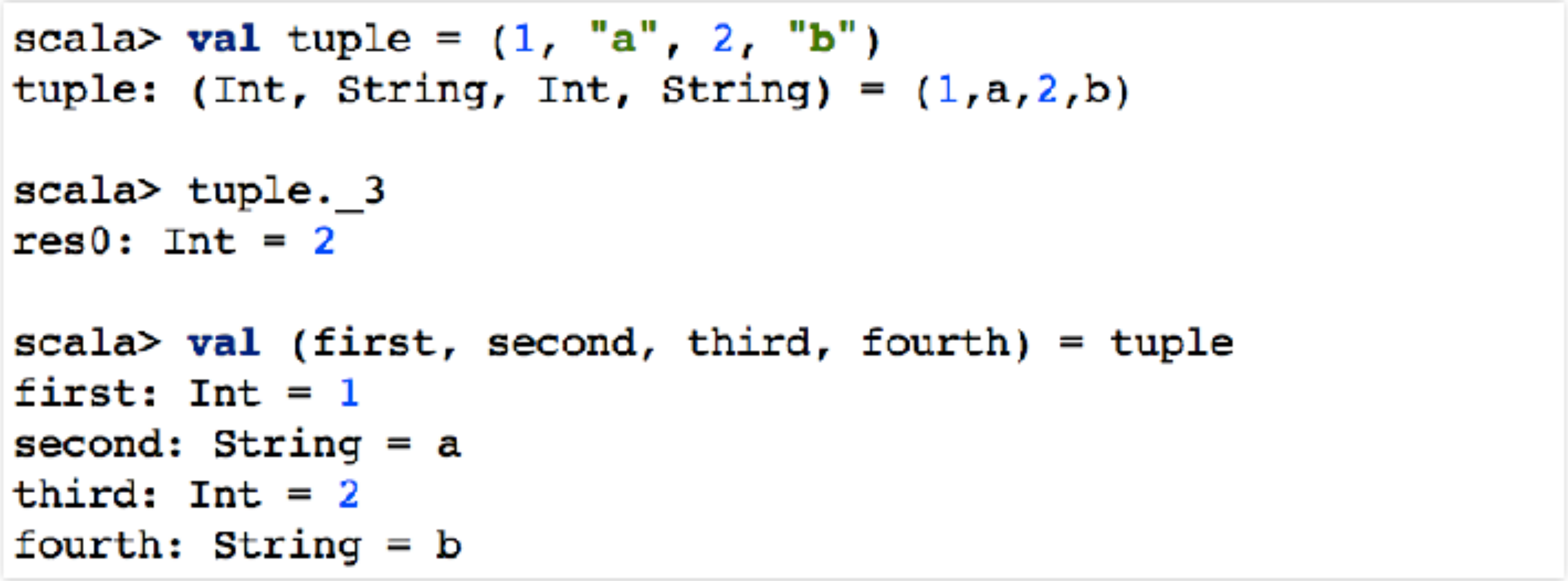
Pattern Matching is Flexible

- You can match on many different kinds of values
  - Literal values, like “12:00”
  - Use guard conditions to be more specific
  - Match on only some parts of a value
  - More specific cases must come first, more general last
  - If you use the _ or a simple name with no type, both match on everything

Exhaustiveness

- When you see the `case` keyword, pattern matching is in play
- Case classes and ADTs provide compile-time exhaustiveness checking that all possible conditions have been have been met

Pattern Matching Tuple Values



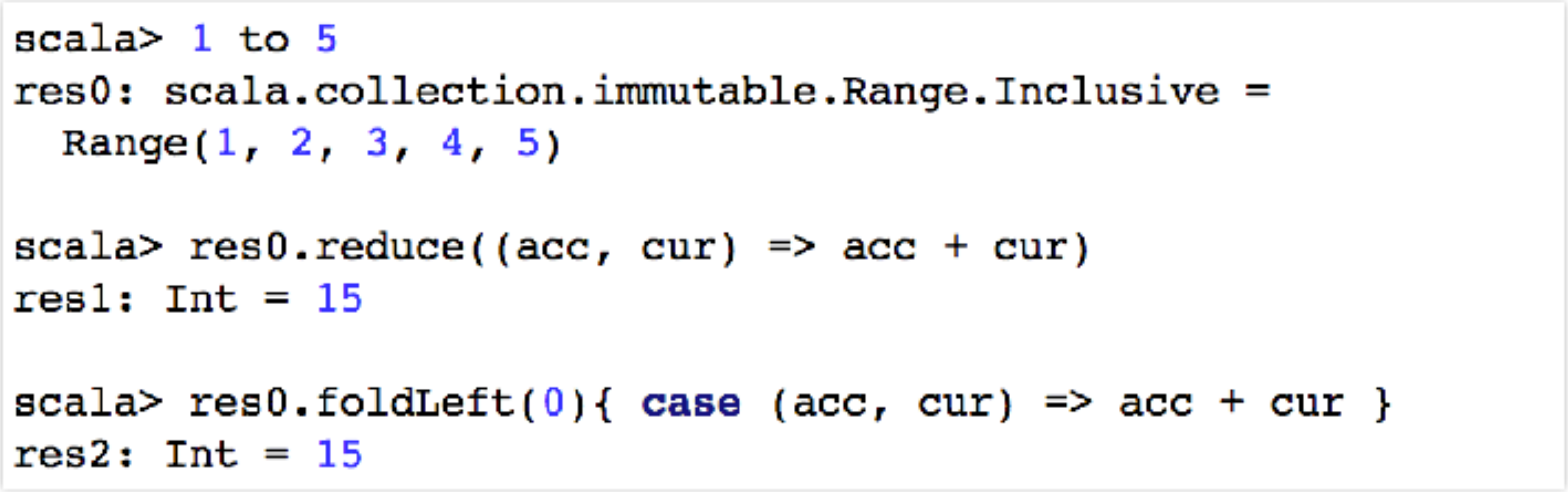
```
scala> val tuple = (1, "a", 2, "b")
tuple: (Int, String, Int, String) = (1,a,2,b)

scala> tuple._3
res0: Int = 2

scala> val (first, second, third, fourth) = tuple
first: Int = 1
second: String = a
third: Int = 2
fourth: String = b
```

Typesafe

Pattern Matching HOF Arguments



```
scala> 1 to 5
res0: scala.collection.immutable.Range.Inclusive =
  Range(1, 2, 3, 4, 5)

scala> res0.reduce((acc, cur) => acc + cur)
res1: Int = 15

scala> res0.foldLeft(0){ case (acc, cur) => acc + cur }
res2: Int = 15
```

Typesafe

Lesson Summary

- Having completing this lesson, you should be able to:
  - Describe how to use pattern matching to handle different values in different ways
  - Outline how case classes and ADTs help in pattern matching
  - Illustrate how to extract values from tuples

Typesafe