СОПОСТАВЛЕНИЕ С ОБРАЗЦОМ

(продолжение)

КОЛЛЕКЦИИ

```
def sum(xs: List[Int], start: Int = 0 ): Int  = xs match {
   case x :: rest => sum(rest, start + x)
   case Nil       => start
}
```

КОЛЛЕКЦИИ

```
def sum(xs: List[Int], start: Int = 0 ) = xs match {
   case List() => start
   case List(x) => start + x
   case List(x, y) => start + x + y
   case List(x, y, z) => start + x + y + z
   case _ => throw new Exception("too many elements")
}
```

КОЛЛЕКЦИИ

```
def sum(xs: List[Int], start: Int = 0 ): Int  = xs match {
   case List() => start
   case List(x, rest@_*) => sum(rest.toList, start + x)
}
```

КОЛЛЕКЦИИ

```
def sum(xs: Seq[Int], start: Int = 0 ): Int = xs match {
   case Seq() => start
   case Seq(x, rest@_*) => sum(rest, start + x)
}
```

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

```
val address = "\\w+, \\w+".r
```

```
def isAddress(string: String): Boolean = string match {
    case address() => true
    case _         => false
}
```

```
isAddress("Volgograd, Russia") == true
```

```
isAddress("Soviet Union") == false
```

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

```
val address = "(\\w+), (\\w+)".r
case class Address(country: String, city: String)
```

```
def readAddress(string: String): List[String] =
  string match {
    case address(city, country) =>
      Some(Address(country, city))
    case _                      => None
  }
```

```
readAddress("Volgograd, Russia") ==
 Some(Address("Russia","Volgograd"))
```

```
readAddress("Soviet Union") == None
```

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

```
val address = "(\\w+), (\\w+)".r
```

```
def readAddress(string: String): List[String] =
  string match {
    case address(parts@_*) => parts.toList
    case _                 => List()
  }
```

```
readAddress("Volgograd, Russia") ==
  List(Volgograd, Russia)
```

```
readAddress("Soviet Union") == List()
```

КОМБИНАЦИЯ

```
val regex = "(Russia|Japan)".r
case class Address(country: String, city: String)
```

```
def getInfo(address: Address): String =
  address match {
    case Address(regex(country), city) =>
      s"$city, $country"
    case Address(_, _) => "unknown country"
  }
```

КОМБИНАЦИЯ

```
val regex = "(Russia|Japan)".r
case class Address(country: String, city: String)
```

```
def getInfo(address: Address): String =
  address match {
    case Address(country@regex(_), city) =>
      s"$city, $country"
    case Address(_, _) => "unknown country"
  }
```

Мы изучили сопоставление с образцом

В следующем разделе изучим частичные функции