Scala Basics Tour - Collections

Kason Chan

This talk was compiled with tut

```
val helloWorld = "Hello World!"
// helloWorld: String = Hello World!

println(helloWorld)
// Hello World!
```

Agenda

- Tuples
- Collections
  - List
  - Set
  - Map

Tuples

- Finite list (sequence) of elements so that they can be passed around as a whole
- Heterogeneous - hold different types

```
val pair = ('name, 'id)
// pair: (Symbol, Symbol) = ('name,'id)

pair.swap
// res1: (Symbol, Symbol) = ('id,'name)

val tuples = (1, "Hello World!", true, 'aSymbol)
// tuples: (Int, String, Boolean, Symbol) = (1,Hello World!,true,'

tuples._1
// res2: Int = 1

tuples._4.name
// res3: String = aSymbol

tuples.productIterator.foreach(print)
// 1Hello World!true'aSymbol
```

Collections - List

- Finite list (sequence) of elements so that they can be passed around as a whole
- Homogeneous - hold different types
- Construct a list
- Take the length of a list
- Reverse a list

```
val bc = 'b :: 'c :: Nil
// bc: List[Symbol] = List('b, 'c)

val defg = List('d, 'e, 'f, 'g)
// defg: List[Symbol] = List('d, 'e, 'f, 'g)

val abcdefgh = 'a :: bc ::: (defg :+ 'h)
// abcdefgh: List[Symbol] = List('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h)

abcdefgh.length
// res5: Int = 8

abcdefgh.reverse
// res6: List[Symbol] = List('h, 'g, 'f, 'e, 'd, 'c, 'b, 'a)
```

Collections - List

- Accessing the end of a list:

```
abcdefgh.head
// res7: Symbol = 'a

abcdefgh.headOption
// res8: Option[Symbol] = Some('a)

List().headOption
// res9: Option[Nothing] = None

abcdefgh.tail
// res10: List[Symbol] = List('b, 'c, 'd, 'e, 'f, 'g, 'h)

abcdefgh.init
// res11: List[Symbol] = List('a, 'b, 'c, 'd, 'e, 'f, 'g)

abcdefgh.last
// res12: Symbol = 'h

abcdefgh.lastOption
// res13: Option[Symbol] = Some('h)
```

Collections - List

- Prefixes and suffixes:

```
val numbers = scala.util.Random.shuffle((1 to 10).toList)
// numbers: List[Int] = List(9, 3, 6, 7, 2, 1, 8, 4, 5, 10)

numbers.sorted
// res14: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

numbers.drop(3)
// res15: List[Int] = List(7, 2, 1, 8, 4, 5, 10)

numbers.dropRight(2)
// res16: List[Int] = List(9, 3, 6, 7, 2, 1, 8, 4)

numbers.take(2)
// res17: List[Int] = List(9, 3)

numbers.takeRight(1)
// res18: List[Int] = List(10)
```

Collections - List

- Prefixes and suffixes:

```
val numbers = scala.util.Random.shuffle((1 to 10).toList)
// numbers: List[Int] = List(5, 3, 4, 2, 8, 10, 1, 9, 7, 6)

numbers.splitAt(5)
// res19: (List[Int], List[Int]) = (List(5, 3, 4, 2, 8),List(10, 1

(numbers.take(5), numbers.drop(5))
// res20: (List[Int], List[Int]) = (List(5, 3, 4, 2, 8),List(10, 1
```

Collections - List

- Element selection:

```
abcdefgh.apply(2)
// res21: Symbol = 'c

abcdefgh(2)
// res22: Symbol = 'c

abcdefgh.drop(2).head
// res23: Symbol = 'c

abcdefgh.indices
// res24: scala.collection.immutable.Range = Range 0 until 8
```

Collections - List

- List creation
- Flattening a list of lists

```
val numbers = (1 to 10 by 2).toList
// numbers: List[Int] = List(1, 3, 5, 7, 9)

val letters = ('a' to 'z' by 3).toList
// letters: List[Char] = List(a, d, g, j, m, p, s, v, y)

letters.foreach(print); println
// adgjmpsvy

val listOf5 = List(List(1,2), List(3), List(), List(4, 5)).flatter
// listOf5: List[Int] = List(1, 2, 3, 4, 5)
```

Collections - List

- Zippings lists

```
letters.zipWithIndex
// res26: List[(Char, Int)] = List((a,0), (d,1), (g,2), (j,3), (m,

letters.indices.zip(letters)
// res27: scala.collection.immutable.IndexedSeq[(Int, Char)] = Vec

val zipped = letters.zip(numbers)
// zipped: List[(Char, Int)] = List((a,1), (d,3), (g,5), (j,7), (m

zipped.unzip
// res28: (List[Char], List[Int]) = (List(a, d, g, j, m),List(1, 3
```

Collections - List

- Displaying lists

```
numbers.toString
// res29: String = List(1, 3, 5, 7, 9)

numbers.mkString("<", ",", ">")
// res30: String = <1,3,5,7,9>
```

Collections - List

- Higher-order methods to map over lists:

```
val numbers = (1 to 10 by 2).toList
// numbers: List[Int] = List(1, 3, 5, 7, 9)

val fps = List('fsharp, 'scala, 'haskell, 'scheme, 'clojure, 'elix
// fps: List[Symbol] = List('fsharp, 'scala, 'haskell, 'scheme, 'c

val fpList = fps.map(_.name.toList)
// fpList: List[List[Char]] = List(List(f, s, h, a, r, p), List(s,

fpList.flatten
// res31: List[Char] = List(f, s, h, a, r, p, s, c, a, l, a, h, a,

fps.flatMap(_.name.toList)
// res32: List[Char] = List(f, s, h, a, r, p, s, c, a, l, a, h, a,
```

Collections - List

- Filter lists:

```
numbers.filter(_ % 3 == 0)
// res33: List[Int] = List(3, 9)

fps.filter(_.name.contains('l'))
// res34: List[Symbol] = List('scala, 'haskell, 'clojure, 'elixir)

numbers.partition(_ % 3 == 0)
// res35: (List[Int], List[Int]) = (List(3, 9),List(1, 5, 7))

(numbers.filter(_ % 3 == 0), numbers.filter(_ % 3 != 0))
// res36: (List[Int], List[Int]) = (List(3, 9),List(1, 5, 7))

fps.partition(!_.name.contains('l'))
// res37: (List[Symbol], List[Symbol]) = (List('fsharp, 'scheme),L

(fps.filter(!_.name.contains('l')), fps.filter(_.name.contains('l'
// res38: (List[Symbol], List[Symbol]) = (List('fsharp, 'scheme),L
```

Collections - List

- Filter lists:

```
val numbers = (1 to 10 by 2).toList
// numbers: List[Int] = List(1, 3, 5, 7, 9)

val fps = List('fsharp, 'scala, 'haskell, 'scheme, 'clojure, 'elix
// fps: List[Symbol] = List('fsharp, 'scala, 'haskell, 'scheme, 'c

numbers.takeWhile(_ < 4)
// res39: List[Int] = List(1, 3)

fps.takeWhile(_.name.contains('l'))
// res40: List[Symbol] = List()

numbers.dropWhile(_ < 4)
// res41: List[Int] = List(5, 7, 9)

fps.dropWhile(_.name.contains('s'))
// res42: List[Symbol] = List('clojure, 'elixir)
```

Collections - List

- Filter lists:

```
val numbers = (1 to 10 by 2).toList
// numbers: List[Int] = List(1, 3, 5, 7, 9)

val fps = List('fsharp, 'scala, 'haskell, 'scheme, 'clojure, 'elix
// fps: List[Symbol] = List('fsharp, 'scala, 'haskell, 'scheme, 'c

numbers.span(_ < 4)
// res43: (List[Int], List[Int]) = (List(1, 3),List(5, 7, 9))

(numbers.takeWhile(_ < 4), numbers.dropWhile(_ < 4))
// res44: (List[Int], List[Int]) = (List(1, 3),List(5, 7, 9))

fps.span(_.name.contains('l'))
// res45: (List[Symbol], List[Symbol]) = (List(),List('fsharp, 'sc

(fps.takeWhile(_.name.contains('l')), fps.dropWhile(_.name.contair
// res46: (List[Symbol], List[Symbol]) = (List(),List('fsharp, 'sc
```

Collections - List

- Predicates over lists:

```
val numbers = (1 to 10 by 2).toList
// numbers: List[Int] = List(1, 3, 5, 7, 9)

val numbersLists = List(numbers, List.fill(5)(0))
// numbersLists: List[List[Int]] = List(List(1, 3, 5, 7, 9), List(

numbersLists.exists(_.forall(_ == 0))
// res47: Boolean = true
```

Collections - List

- Folding lists:

```
val numbers = (1 to 10 by 2).toList
// numbers: List[Int] = List(1, 3, 5, 7, 9)

numbers.fold(0)(_ + _)
// res48: Int = 25

numbers.sum
// res49: Int = 25

numbers.foldLeft(0)(_ + _)
// res50: Int = 25

numbers./:(0)(_ + _)
// res51: Int = 25

numbers.foldRight(0)(_ + _)
// res52: Int = 25

numbers.:\(0)(_ + _)
// res53: Int = 25
```

Collections - List

- Create a range:

```
List.range(1, 5)
// res54: List[Int] = List(1, 2, 3, 4)

List.range(1, 9, 3)
// res55: List[Int] = List(1, 4, 7)

List.range(9, 1, -3)
// res56: List[Int] = List(9, 6, 3)
```

Collections - List

- Create uniform lists:

```
List.fill(1)(1)
// res57: List[Int] = List(1)

List.fill(1,2)(1)
// res58: List[List[Int]] = List(List(1, 1))

List.fill(1,2,3)(1)
// res59: List[List[List[Int]]] = List(List(List(1, 1, 1), List(1,

List.fill(1,2,3,4)(1)
// res60: List[List[List[List[Int]]]] = List(List(List(List(1, 1,

List.fill(1,2,3,4,5)(1)
// res61: List[List[List[List[List[Int]]]]] = List(List(List(List(
```

Collections - List

- Create uniform lists:

```
List.tabulate(5)(x => x + x)
// res62: List[Int] = List(0, 2, 4, 6, 8)

List.tabulate(5,3)(_ + _)
// res63: List[List[Int]] = List(List(0, 1, 2), List(1, 2, 3), Lis
```

- Processing multiple lists

```
(List(10, 20), List(3,4,5)).zipped.map(_ + _)
// res64: List[Int] = List(13, 24)
```

Collections - Set

```
val ranks = (2 to 10).map(x => Symbol(x.toString)).toSet + 'ace +
// ranks: scala.collection.immutable.Set[Symbol] = Set('8, '4, '9,

var fruits = Set('apple, 'banana)
// fruits: scala.collection.immutable.Set[Symbol] = Set('apple, 'b

fruits += 'pear

fruits
// res66: scala.collection.immutable.Set[Symbol] = Set('apple, 'ba

import scala.collection.mutable
// import scala.collection.mutable

val snacks = mutable.Set('chips, 'chocolate)
// snacks: scala.collection.mutable.Set[Symbol] = Set('chocolate,

snacks += 'popcorn
// res67: snacks.type = Set('chocolate, 'popcorn, 'chips)

snacks
// res68: scala.collection.mutable.Set[Symbol] = Set('chocolate, '
```

Collections - Map

```
val numbers = Map(1 -> 'one, 2 -> 'two, 3 -> 'three)
// numbers: scala.collection.immutable.Map[Int,Symbol] = Map(1 ->

numbers(3)
// res69: Symbol = 'three

numbers(4)
// java.util.NoSuchElementException: key not found: 4
//   at scala.collection.immutable.Map$Map3.apply(Map.scala:156)
//   ... 923 elided

numbers.getOrElse(4, 'four)
// res71: Symbol = 'four

import scala.collection.mutable
// import scala.collection.mutable

val months = mutable.Map[Int, Symbol](1 -> 'Jan, 2 -> 'Feb, 3 -> '
// months: scala.collection.mutable.Map[Int,Symbol] = Map(2 -> 'Fe

months += (4 -> 'April)
// res72: months.type = Map(2 -> 'Feb, 4 -> 'April, 1 -> 'Jan, 3 -
```

More Collections

- Checkout https://docs.scala-lang.org/overviews/collections/introduction.html

References

- Programming in Scala
- Essential Scala
- Scala Design Patterns

Thank you

- Q&A/Comments/Suggestions?