Collections: Iterate, Reduce, Map

Collections: List, Sets, Maps

- Scala has a high-performance, object-oriented, and type-parameterized collections
  - We already discuss the basic features of some of them
- Scala's collections have higher-order operations like map, filter, and reduce that make it possible to manage and manipulate data with short and expressive expressions
- The root of all iterable collections, Iterable, provides a common set of methods for iterating through and manipulating collection data

2

Collections: Iterate, Map, Reduce

- Scala's collections use higher-order functions extensively to
  - Iterate
    - Usually you will not use a for loop to iterate over a collection
  - Map
    - convert a list item-by-item to a different list
  - Reduce
    - fold a list into a single element
  - and perform a wide range of other operations

3

foreach

- foreach() is a higher-order functions of collections
- It takes a function (a procedure, to be precise) and invokes it on every item in the list
  - It considers one item at a time and executes the specified procedure on it
  - It returns nothing because a procedure is invoked

4

foreach: Example

```
val colors = List("red", "green", "blue")
/* Print on the console the content the list */
colors.foreach( (c: String) => println(c) )
----------------------------------------------------
val colors = List("red", "green", "blue")
colors.foreach( println(_) )
```

5

map

- map() is a higher-order functions of collections
- It takes a function and applies the function on each element of the collection
- The returned values are "stored" in a new collection
  - The type of the new collection can be different from the one of the former one

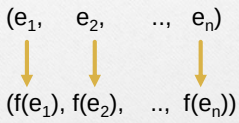
6

map

- map() is used to build one collection from another one
- The new collection has the same number of elements of the one on which map is invoked
- Each element of the new collection has been obtained by applying the specified function on one element of the initial collection

7

map



8

map: Example

```
val numbers = List(32, 95, 24, 21, 17)
val numberTimes2 = numbers.map( (n: Int) => n*2 )
-----------------------------------------------------------------
val numbers = List(32, 95, 24, 21, 17)
val numberTimes2 = numbers.map( _*2 )
```

9

map: Example #2

```
val colors: List[String] = List("red", "green", "blue")
/* The returned list contains the lengths of the strings
of the "input" list.
The data type of the two lists is not the same */
val sizes: List[Int] = colors.map( (c: String) => c.size )
```

10

reduce

- reduce() is a higher-order functions of collections
- It takes a function that combines two list elements into a single element
- It "reduces" the list given a reduction function, starting with the first element in the collection
  - i.e., it returns a single value that is obtained by applying the specified function over one pair of elements (of the collection) at a time

11

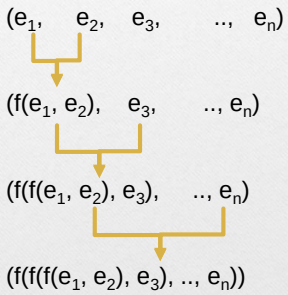
reduce

- Given a list l and a reduction function f, reduce() operates as follows

1. Apply f on the first two elements of the list l
2. Substitute the first two elements of the list l with the value obtained at Step 1
3. If the updated version of l contains one single element then return that element as the final output of the reduce operation. Otherwise go to Step 1

12

reduce



13

reduce

- If the applied function is commutative and associative, the order of the elements in the collection is not relevant
- The reduce function is applied on two elements of the same type and return a new value of the same type

14

reduce: Example

```
val numbers = List(32, 95, 24, 21, 17)
val sum = numbers.reduce( (n1: Int, n2: Int) => n1+n2 )
----------------------------------------------------------------
val numbers = List(32, 95, 24, 21, 17)
val sum = numbers.reduce( _ + _)
```

15

reduce: Example #2

```
val colors: List[String] = List("red", "green", "blue")
/* Concatenate the strings of the "input" list. */
val concatenated = colors.reduce( _ + _)
```

16

Other mapping operations

Name	Example	Description
collect	List(0, 1, 0).collect({case 1 => "ok"})	Transforms only the matched elements
flatMap	List("milk,tea","apple") .flatMap.(_.split(','))	Transforms each element using the given function and "flattens" the list of results into this list
map	List("milk","tea").map(_.toUpperCase)	Transforms each element using the given function

17

Other reducing operations

Name	Example	Description
reduce	List(4, 5, 6).reduce(_ + _)	Reduces the list given a reduction function, starting with the first element in the list
reduceLeft	List(4, 5, 6).reduceLeft(_ + _)	Reduces the list from left to right given a reduction function, starting with the first element in the list
reduceRight	List(4, 5, 6).reduceRight(_ + _)	Reduces the list from right to left given a reduction function, starting with the first element in the list

18

Other reducing (folding) operations

Name	Example	Description
fold	List(4, 5, 6).fold(0)(_ + _)	Reduces the list given a starting value and a reduction function
foldLeft	List(4, 5, 6).foldLeft(0)(_ + _)	Reduces the list from left to right given a starting value and a reduction function
foldRight	List(4, 5, 6).foldRight(0)(_ + _)	Reduces the list from right to left given a starting value and a reduction function

19

Other reducing (scanning) operations

Name	Example	Description
scan	List(4, 5, 6).scan(0)(_ + _)	Takes a starting value and a reduction function and returns a list of each accumulated value
scanLeft	List(4, 5, 6).scanLeft(0)(_ + _)	Takes a starting value and a reduction function and returns a list of each accumulated value from left to right
scanRight	List(4, 5, 6).scanRight(0)(_ + _)	Takes a starting value and a reduction function and returns a list of each accumulated value from right to left

20

Math and Boolean reduction operations

- There are some particular reducing operations
  - Math reduction operations
  - Booleans reduction operations
- They can be implemented by using the standard reduce operations, or by combing reduce and map operations
- However, since they are common, they are already provide by Scala

21

Math reduction operations

Name	Example	Description
max	List(41, 59, 26).max	Finds the maximum value in the list
min	List(10.9, 32.5, 4.23, 5.67).min	Finds the minimum value in the list
product	List(5, 6, 7).product	Multiplies the numbers in the list
sum	sum List(11.3, 23.5, 7.2).sum	Sums up the numbers in the list

22

Boolean reduction operations

Name	Example	Description
Contains	List(34, 29, 18).contains(29)	Checks if the list contains this element
endsWith	List(0, 4, 3).endsWith(List(4, 3))	Checks if the list ends with a given list
exists	List(24, 17, 32).exists(_ < 18)	Checks if a predicate holds true for *at least one* element in the list
forall	List(24, 17, 32).forall(_ < 18)	Checks if a predicate holds true for *every* element in the list
startsWith	List(23, 4, 3).startsWith(List(23))	Tests whether the list starts with a given list

23

Other operations on lists

Name	Example	Description
distinct	List(3, 5, 4, 3, 4).distinct	Returns a version of the list without duplicate elements
filter	List(23, 8, 14, 21).filter(_ > 18)	Returns elements from the list that pass a true/false function
partition	List(1, 2, 3, 4, 5).partition(_ < 3)	Groups elements into a tuple of two lists based on the result of a true/false function
reverse	List(1, 2, 3).reverse	Reverses the list
drop	List('a', 'b', 'c', 'd').drop(2)	Subtracts the first *n* elements from the list

24