# Grouping Things

Grouping, Combining, Sorting.

## A Couple of Tricks

You can create Vectors with a multi-line format, for easier reading:

val v: Vector[String] = Vector(  
 "dog",  
 "cat",  
 "fish"  
)

This is fine for a script or program, but it won’t work in the Console. **Except** if you enter “:paste mode”:

scala> :paste

If you type that into your console, you can paste multi-line commands in. Type ctrl-d to execute.

## Tuples

### Defining Tuples

Sometimes you want to combine two or more values into a single value.

*Example*: I have a String, but I want to attach to it a sequence-number.

This is a job for a **tuple**:

// Create one Tuple  
  
val t: ( String, Int ) = ( "The turtle lives 'twixt plated decks", 1)  
  
val vt: Vector[ ( String, Int ) ] = Vector(  
 ( "The turtle lives 'twixt plated decks", 1),  
 ( "That practically conceal its sex.", 2),  
 ( "I think it clever of the turtle", 3),  
 ( "In such a fix, to be so fertile.", 4)  
)

You can create Tuples with two, three, or any number of elements (but this gets confusing, and there are better ways to handle complex structures).

scala> val t: ( String, Int, Boolean ) = ( "dog", 3, true )

### Accessing Tuples

The syntax for getting at the parts of a Tuple are pretty straightforward:

// Create a Tuple  
val t: ( String, Int ) = ( "The turtle lives 'twixt plated decks", 1)  
  
// Access its parts  
val t\_stringPart: String = t.\_1  
val t\_intPart: Int = t.\_2

## Grouping

This is useful for many kinds of analysis, particularly for creating *histograms*.

val v: Vector[ Char ] = Vector( 't','h','e','t','u','r','t','l','e','l','i','v','e','s','t','w','i','x','t' )  
  
val g = v.groupBy( i => i )

The syntax for groupBy() is not obvious. Basically, this says, “group this vector by its values.” (If you were to use groupBy() with more complex structures, the syntax would actually make *more* sense.)

The result of the above v.groupBy( i => i ) is a Map[ Int, Vector[Int] ].

We don’t want to get into Map right now, so let’s turn it into a Vector:

val v: Vector[ Char ] = Vector( 't','h','e','t','u','r','t','l','e','l','i','v','e','s','t','w','i','x','t' )  
  
val g: Vector[ (Char, Vector[Char]) ] = v.groupBy( i => i ).toVector

Now we can look at it: GroupBy took each *distinct* value in the Vector and gathered *all* instances of that value under it. So we get…

* A Vector, with one item for each *distinct* value in the original Vector
* Each item is a Tuple, ( Char, Vector[Char] )

We can examine it:

scala> g.head  
scala> g(0)  
scala> g(1)  
scala> g(3).\_1  
scala> g(3).\_2

## From GroupBy to Histogram

A “Histogram” is simply a list of “value + number-of-occurances”.

We’ve made a data-structure that is “character + list-of-occurances-of-that-character”.

How can we turn this into a histogram?

We want to do *something to everything* in our Vector[(Char,Vector[Char])], so we need a .map():

val charHisto: Vector[ ( Char, Int )] = g.map( t => ( t.\_1, t.\_2.size ))

Another view of the same thing:

val charHisto: Vector[ ( Char, Int )] = {  
 g.map( t => {  
 val newTuple = ( t.\_1, t.\_2.size )  
 newTuple  
 })  
}

What just happened?

* We mapped g, calling each element t.
* For each t, we made a new Tuple, consisting of t.\_1 (which is the Char), and the *size* of t.\_2, that is, how many instances of that character there were.
* The result is a Vector of Tuples, each consisting of a Character, and an Integer: Vector[ ( Char, Int )].
* And that is a histogram.

## Undoing and Redoing Vectors

We went to a lot of trouble to make a text into a Vector[String]:

val v: Vector[String] = Vector(  
 "dog",  
 "cat",  
 "fish"  
)

**Or, in your script…**

val myLines: Vector[String] = loadFile("text/Aristotle\_Politics.txt")

This is useful for a *lot* of things, but for a character-histogram, we want to work with “one big String.” Scala has this covered:

val myLines: Vector[String] = loadFile("text/Aristotle\_Politics.txt")  
val oneBigString: String = myLines.mkString(" ")

The parameter on .mkString("x") says, “jam every element of this collection together, sticking ‘x’ between them.” Some useful values as params for .mkString() are:

* .mkString(" ") (stick a space between elements)
* .mkString("\n") (stick a return-character between them)
* .mkString (stick nothing between them)

### From a Vector of Lines to a Vector of Characters

Undo one Vector and make another:

val myLines: Vector[String] = loadFile("text/Aristotle\_Politics.txt")  
val oneBigString: String = myLines.mkString(" ")  
val myChars: Vector[Char] = oneBigString.toVector  
val myBetterChars: Vector[String] = myChars.map( \_.toString )

What’s with the last line above? Char is boring and limited; Scala’s String class has many more features. So why not take our Vector[Char] and turn it into a Vector[String] (even if each String consists only of one character)?

### And a Little Clean-Up

Char is boring, and so are spaces, so let’s get rid of all the space-characters in our Vector:

val noSpaceVec: Vector[String] = myBetterChars.filter( \_ != " ")

## Make your Character-Histogram!

No help… just do it. You have everything you need.

## Seeing the Results

Visualization of data is an infinitely deep field. Here’s a quick-and-dirty way to get something useful.

val someHisto: Vector( String, Int ) = …  
for ( h <- someHisto) println( s"${h.\_1}\t${h.\_2}" )

The \t means “tab-character”. We’ve just asked Scala to spit out to the Console every element in someHisto, printing the String part, then a tab-character, then the Int part.

You can copy the resulting data, and paste it into *any* spreadsheet application (Excel, Numbers), and use Someone-Else’s-Programming to do you visualization.