

Slide 1

Administrivia

- Reminder: Quiz 4 Thursday. Will not include material from chapter 8.
- Reminder: Homework 4 on Web, due Friday.

Slide 2

Collection Methods — Summary

- Scala offers many, many ways to operate on elements of a collection. Programs that use them are apt to be compact but not necessarily easy to understand right away.
- (Finish encryption example from last time, quickly. Notice by the way that we did not need `toCharArray` — most of the collection methods can be applied directly to `Strings`. For the record, though, it's in the API for `RichString`.)

One More Construct — Loops

Slide 3

- With what we have already we can build programs to do pretty much anything programs can do — but sometimes in a way that seems a little forced.
- Like arrays and lists, loops give you one more thing in your “bag of tricks” and can sometimes produce code that’s simpler and easier to understand. Functional languages tend to use recursion to achieve iteration; imperative languages tend to use loops.

while and do while Loops in Scala

Slide 4

- These loops repeat a statement or block (the *loop body*) while some condition (the *loop condition*) is true. One variant (*while*) tests the loop condition before each repetition; the other (*do while*) tests after each repetition. Normally the loop body contains something that moves to the next iteration.
- Simple example (prints values 0 through 9):

```
var n = 0
while (n < 10) {
  println(n)
  n += 1
}
```
- (Most languages that support loops offer something that looks pretty similar to this.)

while and do while, Continued

Slide 5

- These two forms of loops are particularly useful when you don't know in advance how many times you need to execute the loop body. (So, the simple example is not really a good one.)
- However, they have some drawbacks — such as what happens when you forget to put something in the loop body that moves to the next iteration. So ...

for Loops in Scala

Slide 6

- These loops let you repeat a statement or block (the loop body) for a sequence of values. Most languages that support loops offer something along similar lines, but it may be significantly less capable.
- Simple examples similar to what most languages support:

```
for (i <- 0 to 9) {  
  println(i)  
}  
for (i <- 0 until 10) {  
  println(i)  
}
```

for, Continued

- In Scala, however, what comes after the `for` and `<-` is actually a form of sequence. (Try typing `0 to 9` in the REPL.)
- So we can also write `for` loops that operate on all values of a collection, for example.
- Multiple generators (see the textbook) allows us to concisely do what might require nested loops in another language.

Slide 7

Examples

- Simple example — summing integers read from standard input, ending with “quit”.
- More complex example — finding primes less than some limit value using the “sieve of Eratosthenes” approach.

Slide 8

Minute Essay

- Try writing a `while` or a `for` loop to print the squares of the numbers from 1 through 10.

Slide 9

Minute Essay Answer

- Here is one way:

```
for (i <- 1 to 10) { println(i*i) }
```

Slide 10