Slide 1

Administrivia

- Quiz 3 Tuesday. Questions will probably be similar to those on Quiz 2 but including material on functions and recursion.
- Reminder: Homework 4 due Tuesday.
- Notice that quiz solutions are on the Web (usually posted shortly after the quiz).
- Homework 1 graded, grades mailed, sample "solution" online (for this assignment, a collection of possibly-useful comments on `vim`).
- Reading assignment for today corrected.
- Minor correction to `countdown` function from last time. (Notice/recall that code from class should mostly show up on the "sample programs" page sometime after class.)

Slide 2

Recursion for Repetition — Review/Recap

- One way to repeat something a fixed number of times, or until some condition is true, is with recursion.
- Examples last time included factorial, "count down". (Notice that we can easily make a function a complete program/script by just adding something to the end to get input from the user. Let's do that for `countdown.scala` from last time.)
- Example in book of using recursion to compute sum of numbers.
- Another example — make our count-out-change program keep asking for input until the user says to quit, rather than doing only one calculation.

Slide 3

One More Conditional Construct — Match

- There are situations in which we have a lot of if/else code testing the same variable against various values. Somewhat repetitive, no? Some languages provide more-compact way to express this (e.g., `switch` in C and Java).
- Scala provides something more general — `match`. Allows matching specified variable with multiple conditions . . .

Slide 4

Match, Continued

- Simple example:

```
val c = readChar
c match {
  case 'a' => println("found a")
  case 'b' => println("found b")
  case _ => println("not a or b")
}
```

This is already more powerful than what some languages provide, in that you can match strings. Much more is possible. Details later.

Slide 5

Arrays and Lists — Preview

- With what we've done so far we have enough tools to compute anything we want to compute.
- However, some things are awkward (repetition), and we don't yet have a convenient way to store many values — something similar to subscripted values in math. (Think about writing some sort of drawing program, one for which our bounding-box function might be useful. Probably you want to somehow store a lot of rectangles or more-general shapes. How?)
- Most programming languages give you a way to represent *collections*. Exactly what you get depends on the language — e.g., C gives you only something quite primitive (but close to what the hardware can do), Java gives you something more abstract/useful, and Scala goes even further.

Slide 6

Minute Essay

- Can you think of anything for which arrays and lists would be helpful?