Slide 1

Administrivia

- (Notice e-mail clarifying policy about homework and deadlines.)

Slide 2

Functions in Scala — Review/Recap

- Why functions? reduce duplication, "manage complexity", allow code reuse.
- Define functions with keyword `def`, name, parameters, return type, body.
  Simple example:

```
def sum(x :  Int, y :  Int) :  Int = x + y
```

- Use functions by giving name, values for parameters in parentheses.
  Examples:

```
val x = sum(10, 20)
println("x plus 1 = " + sum(x, 1))
```

  Notice that values for parameters can be expressions.

Slide 3

Functions, Variable Scope, and Scala

- In many programming languages, every variable has a *scope* — the part of the program within which it has meaning and can be referenced.
- In Scala, the scope of a variable starts with its declaration and continues to the end of the block. Notice that a program might have different variables with the same names and different scopes. Simple example:

```
def printIt(x : Int) { println(x) }
val x = 10
printIt(20)
```

What prints? Why?

Slide 4

Functions — Examples

- Last time we were writing a function to find the bounding box around two rectangles. Finish that, and add code to make a complete program. (Getting input values is another place where we could make good use of a function.)

Slide 5

Function Literals and Higher-Order Functions

- Scala lets you define "literals" for types such as `Int` and `String`. It also lets you define literals for functions. That may seem like a strange thing to do, but . . .
- It also supports "higher-order functions" — functions whose parameters are themselves functions. An example from math is function composition. We will see uses for this in programming later. (Short contrived examples for now.)

Slide 6

Repetition and Recursion — Overview

- Having if/else allows us to do a lot of things we couldn't do before, but there are still things we can't do easily, mostly involving some sort of repetition. Simple example — adding something to the grade program that would prompt for six quiz scores. Another example might be trying to use our bounding-box function to find a bounding box to enclose more than two rectangles, with the choice of how many up to the user.
- Scala provides many ways to do this. We will look at recursion first.

Slide 7

Recursion

- Basic idea of recursion is to solve a problem by defining
  - "base cases" we can solve easily, and
  - a way of reducing other cases to "smaller" instances of the problem
- Simple examples abound in math; a traditional first example is computing the factorial of an integer. We can define $n!$ as the product of the integers from 1 through n, or we can use a recursive definition:

$$n! = \begin{cases} n \cdot (n-1)! & if\ n > 1 \\ 1 & otherwise \end{cases}$$

  This is easy to convert into code in a language that supports recursion . . .

Slide 8

Recursion, Continued

- Key ideas in recursion:
  - One or more base cases that can be solved without recursion.
  - A way of splitting up other cases into one or more *smaller* recursive calls plus some other logic.
- Very important that recursive calls be somehow smaller, so that you eventually reach a base case!
- More examples next time . . .

Slide 9

Minute Essay

- None — quiz.