Slide 1

Administrivia

- Homework 1 grades mailed.
- Reminder: Homework 3 due today.
- Midterm *next* Wednesday.
- Quiz 2 Wednesday. Chapters 3 and 4 only.

  Likely questions are "what does this program do/print?" or "write some code to do some-particular-thing".

  (For the latter, I am less picky about syntax details than the compiler/interpreter.)

  Quiz solutions available on Web after class.
- Homework 4 will be on the Web early tomorrow; due next Monday. (I will send mail.)

Slide 2

More Administrivia

- `vi` tip: If you put the cursor on a parenthesis or brace, `vi` will highlight it and its match, if it has one. If you press%, the cursor moves to the matching brace, again if there is one. Very helpful in diagnosing some kinds of problems!

Slide 3

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 4

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 5

Functions — Examples

- Last time we were writing a function to compute roots of a quadratic equation. What we have is incomplete but good enough to demonstrate defining a function. What do we need to add to make it a complete program/script?
- Another example we could modify to make use of functions: Change-counting program repeats the logic to print value(s) only if nonzero.

Slide 6

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.

Slide 7

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 8

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:

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

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

Slide 9

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!
- As one more example for now — function to "count down" (print numbers from starting point through 1).

Slide 10

Sidebar: Input/Output Redirection

- Normally programs run from the command line write output to the terminal window. Can instead "redirect" output to a file:

  $>$ `outfile` (overwrite)

  $>>$ `outfile` (append)
- Normally programs get input from the keyboard, but can also make them get input from a file with $<$.

  (How could this help you in checking your programs?)
- Finally, can use "pipes" (vertical-bar |) to have output from one program become input to another. Example:

  `ruptime | grep diasw` (show status of Linux-only machines)

  Very powerful idea! this and some other ways of connecting simple programs makes for a very powerful and flexible environment.

Minute Essay

- Are you watching the video lectures? if so, are you finding them helpful? if not, why not?
- What have you found interesting about the programming assignments so far? What has been difficult?

Slide 11