Slide 1

Administrivia

- Homework 1 grades and comments sent by e-mail. (This is how I normally grade programming homework.)
- Reminder: Homework 2 due 11:59pm today.
- Reminder: Quiz 2 Tuesday.

  Questions might be of the form "what does this Scala code do?" or "write some Scala code to do the following task". (For the latter, I am less picky about syntax details than the compiler/interpreter.)
- `vi` tip: If you put the cursor on a parenthesis or brace, `vi` will highlight it and its match, if it has one. Very helpful in diagnosing some kinds of problems!

Slide 2

Minute Essay From Last Lecture

- How programs are laid out / formatted — does it matter? to the compiler/interpreter? to the instructor?
- Mostly the compiler/interpreter does not care about "whitespace". So for example you can write an if/else all on one line or split it across many lines, and the details of where you put the brackets and how much to indent do not matter. (In Python indentation does matter — !)
- Human readers care more. I don't insist on a particular style but encourage you to aim for something relatively readable and consistent.

Slide 3

Functions in Scala — Review/Recap

- 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

Sidebar: User Input in Scala

- We've been using `readInt` to get input from the person running our programs. This is simple, but if the human types something other than a number, including a number preceded by spaces, the program "crashes".
- This is somewhat ugly, but arguably better than what some other programming languages do, which is to provide some sort of error indication that programmers can ignore, in which case the program will probably go wrong in some mysterious way.
- Scala instead uses an *exception* to indicate the error. By default, an exception crashes the program. Programmers can deal with them more gracefully. That may not be covered until POP II — focus for now on programming logic. We might try other solutions when we have more tools . . .

Slide 6

Sidebar: Formatted Output in Scala

- Remember that you can build up strings to print using string concatenation (the "+" operator when one operand is a string) and the fact that Scala knows how to turn most things into `String`s. We don't yet have a way to make floating-point numbers print nicely. Later.
- Or . . . for anyone who has seen `printf` in another language, Scala has that too, e.g.,

```
val x = 2
println("square root of " + x + " = " + math.sqrt(x))
printf("square root of %d = %f", x, math.sqrt(x))
```

and if we replace `%f` in the first parameter to `printf` with `%.2f`, the result prints with only two digits after the decimal point.

Slide 7

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. A simple example for now could be a function that, given an `Int` and a function that maps `Int`s to `Int`s calls the function and prints its input and output in some nice way.

Slide 8

Example — Units Converter

- As an example of a program using multiple functions, we could write one that does various kinds of conversions — inches to centimeters, centimeters to inches, etc.
- We'll have the program first prompt for a character saying which kind of conversion is wanted, then for a number to convert.

Slide 9

Example — Calculating Bounding Boxes

- Something that's of interest in graphics programs is finding the smallest rectangle that encloses one or more other shapes — a "bounding box".
- Let's try writing a function that computes a bounding box for two rectangles. We'll represent rectangles as tuples of four `Int`s — the x and y coordinates of the top left corner and the width and height.
- (Tuples were discussed in chapter 3. Briefly, tuple in Scala are similar to tuples in math — ordered lists of elements, of fixed size. The syntax for making one in Scala is to enclose one or more values in parentheses.)

Slide 10

Preview — Repetition and Recursion

- 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 11

Minute Essay

- What did you find interesting about the problems for Homework 2? What did you find difficult?
- About how long did you spend on Homework 2 (if you know or can estimate)?