Slide 1

Administrivia

- (Everyone turned in Homework 1, right? comments soon!)
- Reminder: Quiz 1 Thursday. Questions based on reading and lectures so far (through today). Likely to focus on material in chapter 3. “Open book/notes”, meaning access to textbook, your notes, anything on course Web site.
- Code from class will be on the course Web site (“sample programs”), sometime after class.
- (Review minute essay from last time. Notice that when there’s an answer it will be in the not-preliminary version of the slides/notes online.)

Slide 2

Scala and Representing Numbers — Review/Recap

- Computer hardware typically represents integers as a fixed number of binary digits, using “two’s complement” idea to allow for representing negative numbers. Scala, like many (but not all!) programming languages bases its notion of integer data on this, but also has a notion of different types with different sizes (e.g., `Int` versus `Long`).
- Hardware also typically supports “floating-point” numbers, with a representation based on a base-2 version of scientific notation. This allows representing not only fractional quantities but also allows representing larger numbers than would be possible with fixed-length integers. Notice that only fractions that can be written with a denominator that’s a power of two can be represented exactly. Again Scala goes along with this and provides two different “sizes” (`Float` and `Double`).

Slide 3

Numbers in Math Versus Numbers in Programming

- The integers and real numbers of the idealized world of math have some properties not (completely) shared by their computer representations.
- Math integers can be any size; computer integers can't.
- Math real numbers can be any size and precision; floating-point numbers can't. Also, some quantities that can be represented easily in decimal can't be represented exactly in binary.
- Math operations on integers and reals have properties such as associativity that don't necessarily hold for the computer representations. (Yes, really!)

Slide 4

Text Data

- Remember that computers represent everything using ones and zeros. How do we then get text? well, we have to come up with some way of "encoding" text characters as fixed-length sequences of ones and zeros — i.e., as small(ish) numbers.
- Several different encodings have been used over the years. One of the earliest schemes was ASCII, which uses 7 bits. That allows for 2^7 (128) characters, which is plenty for numbers, the Roman alphabet, and punctuation and other special characters. Great for English speakers, not so much for others. Unicode originated as a 16-bit encoding, which was thought to be plenty. That turned out not to be true, so Unicode is evolving. (Skim the Wikipedia article to get a sense of what issues are involved.)
- Programming languages make different choices about how to represent characters. Scala's `Char` type is 16-bit Unicode. (Some older languages use ASCII instead.) Single-character literals use single quotes.

Slide 5

Text Data, Continued

- Something else that's needed often enough to be discussed at this point — "strings" of characters.
- Again different programming languages make different choices, but most represent string literals using text contained in double quotes.

  Of course you might then ask how you put a double-quote character in a string! The answer — "escape characters". Described in more detail in textbook.
- Unlike the other types we've talked about (and booleans, which we haven't but which are described in the book), strings are *not* fixed in size. That sort of leads into the next topic . . .

Slide 6

Objects and Methods

- Text strings don't really correspond to anything the hardware can work with as directly as it works with integer and floating-point numbers. So how to represent them is left somewhat more to the discretion of the programming language. They're a simple example of a kind of thing we might want to be able to work with that's somewhat more complicated than what the hardware provides.
- To make working with things other than simple numbers easy, Scala, again like many (but not all!) programming languages has a notion of *objects* (i.e., it is an *object-oriented* (OO) language).
- Remember that we defined a type as a set of values together with some operations on them? In OO-speak, an *object* is something with a value of a particular type, and its *methods* are operations that the type says can be done on it (e.g., arithmetic operations on integers).

Slide 7

Objects and Methods in Scala

- In Scala (unlike some other popular programming languages), everything is an object. This makes some things very convenient (though it puts a certain distance between the language and the hardware, which *may* have negative effects on performance).
- Some operations on objects just do something, without any need for more information (e.g., `toInt` converts a `Double` to an `Int`). Others requre *parameters* (e.g., integer addition).
- Basic syntax for invoking an object's methods requires a period, the name of the method, parentheses, and any parameters. Scala allows many of these to be omitted if it can figure out what you mean. (Indeed, some methods that take no parameters must *not* be followed by parentheses.)

Slide 8

Objects and Methods in Scala

- Many useful "library" methods built into the language. The REPL provides some support in the form of tab completion. (Try some things with integers and strings!)
- Library methods include many for working with text strings, plus the `math` object. See book for details.

Slide 9

Variables

- We know enough — more than enough — at this point to use the Scala REPL as a calculator. But that's not really programming, since if we want to do the same calculation for different sets of values we'd have to retype everything.
- To do almost anything interesting, we need some way to save values and give them names, so we can reference them again. So Scala, like most programming languages, has a notion of *variables*, similar (but not identical!) to variables in math. (The biggest difference is that some Scala variables can take on different values as a calculation proceeds.)
- Basic syntax for defining variables requires a keyword (`val` or `var`), a type, and a name. Can omit type if Scala can guess. `val` versus `var`? Former can't change value, latter can.
- Now we can start writing programs . . .

Slide 10

Minute Essay

- How is the reading coming? is it helpful in supplementing what I say in class, does it make sense, etc.?