An interpreter lurks under every cobblestone.
—sk

Garbage collector! Get thee to a nunnery!
—sk imitating those who like to manage their own memory, adapted from *Hamlet*

1 The Only Language You’ll Ever Need

The only language you’ll ever need is Java. That’s it. End of story. Well, maybe C++ for those who like managing memory. But surely one of those two is all you need, right?

Why are we bothering to study programming languages?

1. Because in the history of programming languages, people, time and time again, have believed that the hot language of the day is the last language that will ever be needed. But history has shown that “the” hot language (the only language you’ll ever need) changes about every 7 years. People create new languages all the time, and chances are, you are going to have to deal with some or many of these.

2. Because, as the project manager of a software company, you will be responsible in making a huge choice for a given project: which language to use. You need to be able to make an informed decision.

3. Because you might have to design a language, and the design choices you make may end up affecting a large number of people. You need to be able to make good choices.

2 Interpreters are Everywhere

A lot of this course is about interpreters. Let’s start out with a concise and simple working definition of interpreter.
Definition 1 (Interpreter) An interpreter is a program that consumes a program and produces an answer.

Examples:

1. GIFs are programs. A GIF interpreter consumes some GIF program and produces an image (the answer). Note that the GIF language has a simple notion of looping—that is, pixels are drawn horizontally (in a loop) and then vertically (in a loop).

2. Animated GIFs are programs. The animated gif language is slightly more complicated than ordinary GIFs. An animated GIF program specifies a series of images to be displayed, along with ways of specifying loops (such as for delays).

3. Flash is an even more sophisticated interpreter.

4. When we type commands into the shell, we are writing programs. For example, *.java is a program, interpreted by the shell, which produces a list of Java files (the answer).

3 Some lofty principles

Principle 1 (Programmibility) Make things programmable (or, make things scriptable).

This is one of the big achievements of the 90s. Make your web pages scriptable. Make Photoshop scriptable. Make your toaster scriptable.

But the utility of the programmibility principle, relies on a more fundamental principle:

Principle 2 (Laziness) A description of how to get an answer is often much shorter than describing the answer itself.

4 Tensions

There are several tensions fundamental to computer science:

1. The tension among entities competing for sparse resources (think economics)

2. The tension between safety and liberty

3. The tension between what is good (enough) and what is great

We will see examples of these tensions throughout the course.
5 What this course will teach you
   1. How to understand big languages
   2. How to implement small languages
   3. Functional programming
   4. To understand important language decisions/trade-offs/differences

We will be using one syntax in the course. This forces you to look beyond syntax (which is often superficial) to the important stuff.

6 A bureaucratic warning

You must check the newsgroup. You must check the newsgroup often. You are responsible for announcements made on the newsgroup. The newsgroup is brown.cs.cs173.