CS3:
Introduction to Symbolic Programming

Lecture 14:
Review
Scheme and other languages

Spring 2008

Nate Titterton
nate@berkeley.edu

Schedule

13	April 21-25	Lecture: Introduction to the big project Advanced lists Lab: Big project – introduction, and choose partners (checkoff #1)
14	April 28 – May 2	Lecture: Advanced lists Scheme vs. other Languages Lab: Big project (checkoff #2)
15	May 5 – 9	Lecture: (guest) CS at Berkeley and outside… Lab: Big project (checkoff #3, due Friday midnight)
16	May 12	Lecture: Final Exam Review Lab: *no thank you!*
	Wed, May 21	Final Exam 12:30-3:30, Bechtel Auditorium

Due dates on the final project

Tues/Wed	Thur/Fri
(April 22/23) **Introduction**	*(April 24/25)* **Checkoff 1**
(April 29/30)	*(May 1/2)* **Checkoff 2**
(May 6/7) **Checkoff 3**	*(May 9th, Friday)* **Due (at midnight)**

Any questions about the project?

How about this `flatten`?

- **Recall `flatten`, which takes a generalized-list and returns a "flat" list. You saw three solutions in lab.**

- **Fill in the blanks below for a fourth solution:**

```
(define (flatten thing)
   (if (list? thing)
      (reduce ________ (map flatten thing))
      (______ thing)))
```

Tree-structured directories (1)

- **From lab:**

  Consider the following Scheme representation for a hierarchical file system. A file is represented by a list we'll call a file entry.

  The file entry for a non-directory file is a two-element list whose first element is the word FILE and whose second word is the name of the file.

  The file entry for a directory is a list whose first two elements are the word DIRECTORY and the name of the directory, and whose remaining elements are file entries for the files within the directory (which may be directories themselves).

  Write and test a procedure named file-list that, given as argument a file entry for a directory, returns a list of names of the non-directory files anywhere in the corresponding directory tree.

Tree-structured directories (2)

- **Example:**

```
(define *fe1*
   (DIRECTORY a
       (DIRECTORY b)
       (FILE c)
       (DIRECTORY d
           (DIRECTORY e
               (FILE g)
               (FILE h)
               (FILE i))
           (FILE f)))

STk> (file-list *fe1*)
(c f g h i)
```

Random and sequential programming

- **Any questions?**

```
(define (better-converse)
  (let ((phrases  '((tell me about yourself)
                    (that is interesting)
                    (you do not say)
                    (tell me more)
                    (please say more about that)
                    (I never heard something like
                     that before!) )))
    (show-line (list-ref phrases
                         (random (length phrases))))
    (let ((what-he-said (read-line)))
      (if (equal? what-he-said '(arggh))
          (show-line '(oh goodbye then))
          (better-converse)) ) ))
```

Scheme versus other (sequential) languages

Side-effects

- **In Functional programming, this return value of a procedure is all that matters**

- **However, computers are asked to do lots of things other than calculate values that could be passed to another procedure.**
  - **Printing to a screen or a printer**
  - **"Connecting" over a network**
  - **Getting user input**
  - **Playing sounds or music**
  - **Saving to a file**

The language Scheme

- **Scheme allows you to ignore tedium and focus on core concepts**
  - **The core concepts are what we are teaching!**
- **Other languages:**
  - **Generally imperative, sequential**
  - **Lots and lots of syntactic structure – instead of parentheses, there are several special symbols.**
  - **Lots and lots of built-in procedures for doing common things**
  - **Object-oriented is very "popular" now**
- **You can do most of this stuff in Scheme**
  - **It just takes a little more work**

CS3 concepts out in the world

- **Scheme/lisp does show up: scripting languages inside applications (emacs, autocad, flash, Orbitz.com, etc.)**
- **Scheme/Lisp is used as a "prototyping" language**
  - **to quickly create working solutions for brainstorming, testing, to fine tune in other languages, etc.**
- **Recursion isn't used directly (often), but recursive ideas show up everywhere**

Java

- **Java is a very popular programming language**
  - **Designed for LARGE programs**
  - **Very nice tools for development**
  - **Gobs of libraries (previous solutions) to help solve problems that you might want solved**

Javascript

- **A language that is run inside a browser (i.e., inside Firefox).**
  - **Special procedures for moving browser windows around, history lists, etc.**
- **A modern language, otherwise:**
  - **Higher order functions, object-oriented. Quite scheme-like, really.**
- **Not related to Java! (just capitalizing on the popularity of the name)**

PHP

- **PHP**
  - **Popular language for web development (combined with a web-server and database)**
  - **Lots of features, but little overall "sense"**
  - **Because programs in PHP execute behind a web-server and create, on the fly, programs in other languages, debugging can be onerous.**