CS3:
Introduction to Symbolic Programming

Lecture 13:
Introduction to the big project,
Lists

Fall 2006

Nate Titterton
nate@berkeley.edu

Schedule

13	April 16-20	Lecture: CS3 Projects, Lists Lab: Begin work on CS3 Big Project Reading: Simply Scheme, chapter 20
14	April 23-27	Lecture: Non-functional programming, lists, project review Lab: Non-functional programming Work on projects
15	Apr 30-May 4	Lecture: CS at Berkeley (guest lecture) Lab: Finish projects (due end of week)
16	May 7	Lecture: Exam review *no more labs!*
	Thursday, May 17	*Final Exam, 5-8pm* *F295 Haas*

Give me suggestions for remain lectures, if you have them…

Midterm #2

- **Will be available Wed/Thursday**
  - **(graded Tuesday night).**

Any questions?

6) **tail-recursive `roman-sum`**
7) **price-is-right**
8) **`last-letter`**
9) **chips, drinks, and gum -- `snack3`**

The Big Project

- **Two possible projects:**
  - **Connect4**
  - **Blocks World**
- **You can, and should, work in partnerships**
- **You will have three weeks to work on this (it is due on the last lab)**
- **Worth 15% of your final grade**

Project Check-offs

- **There are 3 checkoffs**

  You need to do them on time in order to get credit for the project

3. **Tell your TA which project you will do and who you will do it with**
4. **Show your TA that you have accomplished something. S/he will comment.**
5. **Show that you have most of the work done: your TA will run your code.**

Due dates on the final project

Tues/Wed	Thur/Fri
(April 17/18) Introduction	*(April 19/20)* Checkoff 1
(April 24/25)	*(April 26/27)* Checkoff 2
(May 1-2) Checkoff 3	*(May 4, Friday)* Due (at midnight)

Lets see the projects in action

What issues matter

- **Does it work?**
  - **This is a primary grading standard…**

- **Programming style**
- **Data abstractions**
- **Reading specifications carefully**
- **Adequate testing**

Working in partnerships

- **Highly recommended!**
  - For those of you continuing with CS, you'll be doing this for many future projects
- **Won't be faster, necessarily**
  - While you are less likely to get stuck, there will be a lot of communication necessary
- **A big benefit will be with testing**
- **Remember, only one grade is given…**
  - this grade will be the same, whether the project is a solo or a partnership

Data structures

- **The format of data used in these projects in a central feature**
  - **A "data structure" (abstract data type) is a specification of that format. Here, generally, lists of lists (of lists).**
  - **Accessors and constructor allow for *modularity*: letting parts of a program work independently from other parts.**

Functional Programming

- **In CS3, we have focused on programming without *side-effects*.**
  - **All that can matter with a procedure is what it returns**
  - **In other languages, you typically:**
    - **Perform several actions in a sequence**
    - **Set the value of a variable – and it stays that way**
  - **All of this is possible in Scheme.**
- **Both projects involve graphics and/or printing, which is a side-effect.**
  - ***Simply Scheme* chapter 20 is nice summary.**
  - **We'll cover this in the next lecture, and in next Tue/Wed's lab.**

Lists

Lists

- **Lists are containers, like sentences, where each element can be <u>anything</u>**
  - **Including, another list**

```
((beatles 4) (beck 1) ((everly brothers) 2) … )

((california 55) (florida  23) ((new york) 45) )

(#f   #t   #t   #f   #f  …)
```

Sentences(words) vs lists: constructors

cons Takes an element and a list Returns a list with the element at the front, and the list contents trailing	
append Takes two lists Returns a list with the element of each list put together	
list Takes any number of elements Returns the list with those elements	sentence Takes a bunch of words and sentences and puts "them" in order in a new sentence.

Sentences(words) vs lists: selectors

car Returns the first element of the list	first Returns the first word (although, works on non-words)
cdr Returns a list of everything but the first element of the list	butfirst Returns a sentence of everything but the first word (but, works on lists)
	last …
	butlast …

What is the point of `cons`? (1/2)

- **`append` and `list` are generally used to conveniently build lists**

- **`cons` is more closely tied to use in recursion, specifically as the combiner in linear recursion**

What is the point of cons? (2/2)

```
(define (square-all seq)
  (if (empty? seq)
      '()
      (cons (square (first seq))
            (square-all (rest seq)))))

(s-a '(1 2 3)) → (cons 1 (cons 4 (cons 9 '())))
```

Sentences(words) vs lists: HOF

map Returns a list where a func is applied to every element of the input list. Can take multiple input lists.	every Returns a sentence where a func is applied to every element of an input sentence or word.
filter Returns a list where every element satisfies a predicate. Takes a single list as input	keep Returns a sentence or word where every element satisfies a predicate
reduce Returns the value of applying a function to successive pairs of the (single) input list	accumulate Returns the value of applying a function to successive pairs of the input sentence or word
apply Takes a function and arguments, and applies that function to its arguments	…

Example: mapping over a list

Here, we `map` (like `every`) over a list of functions:

```
(map (lambda (proc)
       (proc 4 3))
     '(+ - / * expt))

  ➔ (7  4  4/3  12  64)
```

`map` can take multiple list arguments

```
(map + '(1 2 3) '(100 200 300))
➔(101 202 303)
```

The argument lists have to be the same length

```
(define (palindrome? lst)
  (all-true?
    (map equal? lst (reverse lst))))
```

```
(palindrome?
    '(a m a n a p l a n a c a n a l p a n a m a))
  ➔ #t
```

`apply` (not the same as `accumulate`!)

- **apply takes a function and a list, and calls the function with the elements of the list as its arguments:**

```
(apply + '(1 2 3))

(apply cons '(joe '(bob)))

(apply day-span
       '((january 1) (december 31)))
```