Lecture 14: Lists
Scheme vs. other programming languages
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Apr 24-28</td>
<td>Lecture: Lists, other languages</td>
<td>Lab: Big Project</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHECKOFF #2 – Thur/Fri</td>
</tr>
<tr>
<td>16</td>
<td>May 1-5</td>
<td>Lecture: Guest Lecture: what is CS at UCB?</td>
<td>Lab: Finish up the Project</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHECKOFF #3 – Tue/Wed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Project Due on Fri (at midnight)</td>
</tr>
<tr>
<td>17</td>
<td>May 9-14</td>
<td>Lecture: Review of CS3, solving problems</td>
<td>Lab: NONE (the semester is over)</td>
</tr>
<tr>
<td>18</td>
<td>May 18</td>
<td>Final: Thursday, May 18th</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>12:30 – 3:30, 4 Le Conte</td>
<td></td>
</tr>
</tbody>
</table>
Lists
Lists

• Lists are containers, like sentences, where each element can be anything

  - 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

<table>
<thead>
<tr>
<th>cons</th>
<th>Takes an element and a list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns a list with the element at the front, and the list contents trailing</td>
</tr>
<tr>
<td>append</td>
<td>Takes two lists</td>
</tr>
<tr>
<td></td>
<td>Returns a list with the element of each list put together</td>
</tr>
<tr>
<td>list</td>
<td>Takes any number of elements</td>
</tr>
<tr>
<td></td>
<td>Returns the list with those elements</td>
</tr>
<tr>
<td>sentence</td>
<td>Takes a bunch of words and sentences and puts &quot;them&quot; in order in a new sentence.</td>
</tr>
<tr>
<td><strong>Sentence (words) vs lists: selectors</strong></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------</td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>first</td>
</tr>
<tr>
<td>Returns the first element of the list</td>
<td>Returns the first word</td>
</tr>
<tr>
<td></td>
<td>(although, works on non-words)</td>
</tr>
<tr>
<td>cdr</td>
<td>butfirst</td>
</tr>
<tr>
<td>Returns a list of everything but the</td>
<td>Returns a sentence of everything but the</td>
</tr>
<tr>
<td>first element of the list</td>
<td>first word</td>
</tr>
<tr>
<td></td>
<td>(but, works on lists)</td>
</tr>
<tr>
<td>last</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>butlast</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
### Sentences(words) vs lists: HOF

<table>
<thead>
<tr>
<th></th>
<th>map</th>
<th>every</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns a list where a func is applied to every element of the input list.</td>
<td>Returns a sentence where a func is applied to every element of an input sentence or word.</td>
</tr>
<tr>
<td></td>
<td>Can take multiple input lists.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>filter</td>
<td>Returns a list where every element satisfies a predicate. Takes a single list as input</td>
<td>keep</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reduce</td>
<td>Returns the value of applying a function to successive pairs of the (single) input list</td>
<td>Accumulate</td>
</tr>
</tbody>
</table>
A few other important topics re: lists

2. map can take multiple arguments

4. Association lists

6. Generalized lists
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

Quiz: Can you write all-true? without if and cond?
Association lists

• Used to associate key-value pairs

(((i 1) (v 5) (x 10) (l 50) (c 100) (d 500) (m 1000)))

• assoc looks up a key and returns a pair

(assoc 'c '(((i 1) (v 5) (x 10) ... )))

⇒ (c 100)

;; Write sale-price, which takes a list of items and a
;; table of item-price pairs, and returns a total price
(define *price-list* '(((bread 2.89) (milk 2.33)
                         (cheese 5.21) (chocolate .50)
                         (beer 6.99) (tofu 1.67) (pasta .69)))
Generalized lists

• Elements of a list can be anything, including any list

• Lab materials discuss
  - flatten (3 ways)
  - completely-reverse
  - processing a tree-structured directory
Scheme versus other languages
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; Chapter 20 is a good place to start
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 (built in commands)
  - Object-oriented is very "popular" now
CS3 concepts out in the world

• Scheme/lisp does show up: scripting languages inside applications (emacs, autocad, Flash, 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 and PHP

• 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

• PHP
  - Popular course 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.
SQL resembles HOFs

• SQL if for database retrieval

• query: “Tell me the names of all the lecturers who have been at UCB longer than I have.”

```sql
select name from lecturers
  where date_of_hire <
    (select date_of_hire from lecturers where name = 'titteton');
```

• query: “Tell me the names of all the faculty who are older than the faculty member who has been here the longest.”

```sql
select L1.name from lecturers as L1 where L1.age >
  (select L2.age from lecturers as L2
   where L2.date_of_hire =
     (select min(date_of_hire) from lecturers) );
```
Problems
A list version of electoral-votes

Write a higher-order procedure named electoral-votes which takes a predicate as its single argument. The procedure will sum up the 2008 electoral votes for states that satisfy the predicate.

(electoral-votes california?) ➞ 55
(electoral-votes blue-state?) ➞ 212

The database of states and their electoral votes is in a global variable *states*:

((ca 55) (me 4) (nj 15) ...)

The predicate takes the state's two-letter abbreviated name as its argument. You do not have to write these predicates; rather, you only need to write electoral-votes such that it works properly with any proper predicate.