
CS3:

Introduction to Symbolic Programming

Lecture 13:

Introduction to the big project Lists

Fall 2006

Nate Titterton
nate@berkeley.edu

Schedule

13	Nov 20-24	Lecture: Lists, and introduce the big project Lab: Lists; start on the project
14	Nov 27–Dec 1	Lecture: Lists; Computer Science at UCB Lab: Work on the project
15	Dec 4 – Dec 8	Lecture: Course summary; practice problems Lab: Finish project; due at end of week
	Dec 16 th	Final Exam (Saturday, 8am-11am)

Midterm #2

Any questions?

The Big Project

- **Three possible projects:**
 - **Database**
 - **Yukon**
 - **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
<i>(Nov 21/22)</i> Introduction	<i>(Nov 23-24)</i> Thanksgiving
<i>(Nov 28-29)</i> Checkoff 1	<i>(Nov 30 – Dec 1)</i> Checkoff 2
<i>(Dec 5-6)</i> Checkoff 3	<i>(Dec 8, Friday)</i> Due (at midnight)

Only two more lectures (after this one)...

What would you like to do?

- **Hear about the CS major, and other courses...**
- **Do exam-type problems...**
- **Review...**

**Lets see the
projects in action**

What issues matter

- **Does it work?**
 - **This is a primary grading standard...**
- **Programming style**
- **Reading specifications carefully**
- **Error checking inputs (especially in the database project)**
- **Adequate testing**
- **Code reuse (again, with the database)**

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

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.

Printing, and sequencing

- **With *Blocks World* and *Yukon* you will need to display information.**
 - *Simply Scheme* chapter 20 is nice summary.
 - And, all the projects have file input /output routines that you don't need to "understand", as well as user input routines.

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.

Strings versus words

- **One useful data structure is a string**
 - **Strings are surrounded by double quotes when printed.**
 - **Strings are a native type in Scheme.**
- **In CS3, you used words (sometimes sentences) to present names and other output to the user.**
 - **In the "real world", strings are used.**

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

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

What is the point of cons?

While `append` and `list` make more sense, `cons` is more closely tied to use in recursion:

```
(define (sent-square-all sent)
  (if (empty? sent)
      '()
      (se (square (first sent))
          (sent-square-all (bf sent)))))
```

```
(ssa '(1 2 3)) → (se 1 (se 4 (se 9 '())))
```

```
(define (list-square-all lst)
  (if (null? lst)
      '()
      (cons (square (car lst))
            (list-square-all (cdr lst)))))
```

```
(lsa '(1 2 3)) → (cons 1 (cons 4 (cons 9 '())))
```

Sentences(words) vs lists: selectors

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

Sentences(words) vs lists: HOF

<p>map</p> <p>Returns a list where a func is applied to every element of the input list.</p> <p>Can take multiple input lists.</p>	<p>every</p> <p>Returns a sentence where a func is applied to every element of an input sentence or word.</p>
<p>filter</p> <p>Returns a list where every element satisfies a predicate.</p> <p>Takes a single list as input</p>	<p>keep</p> <p>Returns a sentence or word where every element satisfies a predicate</p>
<p>reduce</p> <p>Returns the value of applying a function to successive pairs of the (single) input list</p>	<p>Accumulate</p> <p>Returns the value of applying a function to successive pairs of the input sentence or word</p>

CS3: **Introduction to Symbolic Programming**

Lecture 13: Introduction to the big project Lists

Fall 2006

Nate Titterton
nate@berkeley.edu

Schedule

13	Nov 20-24	Lecture: Lists, and introduce the big project Lab: Lists; start on the project
14	Nov 27–Dec 1	Lecture: Lists; Computer Science at UCB Lab: Work on the project
15	Dec 4 – Dec 8	Lecture: Course summary; practice problems Lab: Finish project; due at end of week
	Dec 16 th	Final Exam (Saturday, 8am-11am)

Spring 2006 CS3: 2

Midterm #2

Any questions?

The Big Project

- **Three possible projects:**
 - Database
 - Yukon
 - 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 (Nov 21/22) Introduction	Thur/Fri (Nov 23-24) Thanksgiving
(Nov 28-29) Checkoff 1	(Nov 30 – Dec 1) Checkoff 2
(Dec 5-6) Checkoff 3	(Dec 8, Friday) Due (at midnight)

Only two more lectures (after this one)...

What would you like to do?

- **Hear about the CS major, and other courses...**
- **Do exam-type problems...**
- **Review...**

**Lets see the
projects in action**

What issues matter

- **Does it work?**
 - **This is a primary grading standard...**
- **Programming style**
- **Reading specifications carefully**
- **Error checking inputs (especially in the database project)**
- **Adequate testing**
- **Code reuse (again, with the database)**

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

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.

Printing, and sequencing

- With *Blocks World* and *Yukon* you will need to display information.
 - *Simply Scheme* chapter 20 is nice summary.
 - And, all the projects have file input /output routines that you don't need to "understand", as well as user input routines.

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.

Strings versus words

- **One useful data structure is a string**
 - Strings are surrounded by double quotes when printed.
 - Strings are a native type in Scheme.
- **In CS3, you used words (sometimes sentences) to present names and other output to the user.**
 - In the "real world", strings are used.

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

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

Spring 2006 CS3: 17

Some "tips":

With `append`, you erase the middle parentheses

```
(append '( a b c ) ( d (e) f ) )
;;
;;          | |
;;          x x
-> ( a b c d (e) f )
```

With `list`, you add parentheses around the arguments

```
(list '( a b c ) (d (e) f) (g h i) )
;;
;;          | |
;;          v v
-> ( ( a b c ) (d (e) f) (g h i) )
```

With `cons`, the last argument is a list (almost always in the real world, and always in this class). `cons` stretches the opening paren for that second argument to include the first argument:

```
(cons 'a '( b ) )
;;
;;          |
;;          -----
;;          v
-> ( a b )
```

What is the point of cons?

While `append` and `list` make more sense, `cons` is more closely tied to use in recursion:

```
(define (sent-square-all sent)
  (if (empty? sent)
      '()
      (se (square (first sent))
          (sent-square-all (bf sent)))))
```

```
(ssa '(1 2 3)) → (se 1 (se 4 (se 9 '())))
```

```
(define (list-square-all lst)
  (if (null? lst)
      '()
      (cons (square (car lst))
            (list-square-all (cdr lst)))))
```

```
(lsa '(1 2 3)) → (cons 1 (cons 4 (cons 9 '())))
```

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 ...

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