Lecture 10: Fracturing Fractals

cs1120 Fall 2009
David Evans
http://www.cs.virginia.edu/cs1120

Menu

- Problem Set 2
- Mapping Lists
- Problem Set 3

Problem Sets

- Not just meant to review stuff you should already know
  - Get you to explore new ideas
  - Motivate what is coming up in the class
- The main point of the PSs is ***learning***, not ***evaluation***
  - **Don't give up if you can't find the answer in the book** (you won't solve many problems this way)
  - **Do discuss with other students**
  - **Do get help from Help Hours and Office Hours**

PS2: Question 3

Why is

(define (higher-card? card1 card2)
  (> (**card-rank** card1) (**card-rank** card2)

better than

(define (higher-card? card1 card2)
  (> (**car** card1) (**car** card2))

?

Data Abstraction: to understand more complex programs, we need to hide details about how data is represented and think about what we do with it.

PS2: Question 8, 9

- Predict how long it will take
- Identify ways to make it faster

Most of next week and much of many later classes will be focused on how computer scientists **predict** how long programs will take, and on how to **make them faster**.

Question 7 ("Gold Star" answer)

```
(define (find-best-hand hole-cards community-cards)
  (car (sort (possible-hands hole-cards community-cards))
             higher-hand?))
```

How can we do better?

Hmmm....

from last class:

```
(define (pick-minimizer f a b)
  (if (< (cf a) (cf b)) a b))

(define (find-minimizer f p)
  (if (null? (cdr p))
      (car p)
      (pick-minimizer f (car p)
                        (find-minimizer f (cdr p)))))
```

find-best

```
(define (find-best f p)
  (if (null? (cdr p))
      (car p)
      (pick-best f
        (car p)
        (find-best f (cdr p)))))

 (define (pick-best f a b)
  (if (f a b) a b))
```

find-best-hand

```
(define (find-best-hand hole-cards community-cards)
  (find-bestiest
    (possible-hands hole-cards community-cards))
    higher-hand?))
```

Next week: how much better is this?

Mapping Lists

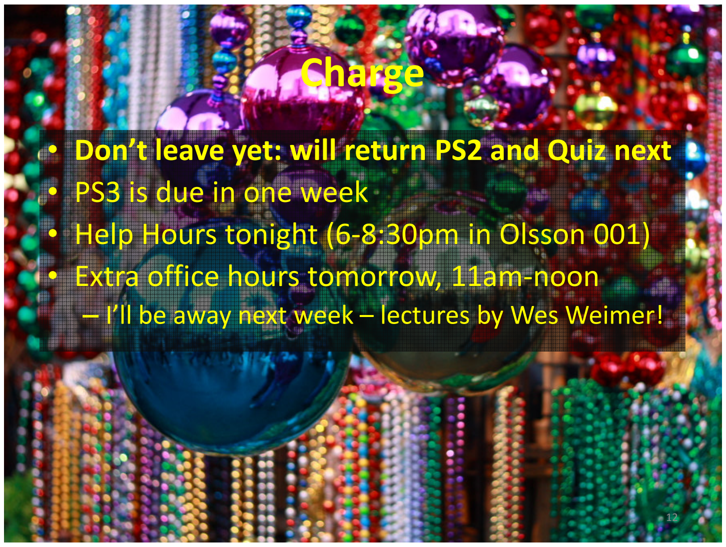
Define a procedure list-map that takes two inputs, a procedure and a list and produces as output a list whose elements are the results of applying the input procedure to each element in the input list. (Example 5.4)

```
> (list-map square (list 1 2 3))
(1 4 9)
> (list-map (lambda (x) (* x 2)) (list 1 2 3 4))
(2 4 6 8)
> (list-map (lambda (x) (if (odd? x) (+ x 1))) (list 1 2 3 4))
(2 2 4 4)
```

list-map

```
(define (list-map f p)
   (if (null? p)
       null
       (cons (f (car p))
             (list-map f (cdr p)))))
```

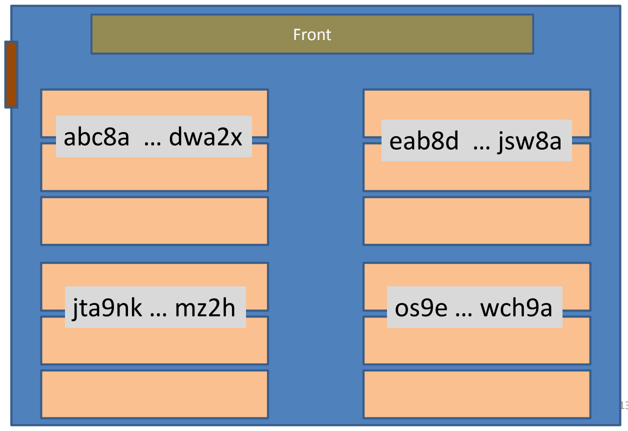
Equivalent to the built-in procedure **map** (except **map** can work on more than one list).



Charge

- **Don't leave yet: will return PS2 and Quiz next**
- PS3 is due in one week
- Help Hours tonight (6-8:30pm in Olsson 001)
- Extra office hours tomorrow, 11am-noon
  – I'll be away next week – lectures by Wes Weimer!

Returning PS2 and Quiz


Front
abc8a ... dwa2x
eab8d ... jsw8a
jta9nk ... mz2h
os9e ... wch9a