Class 23: Objectifying Objects

University of Virginia cs1120

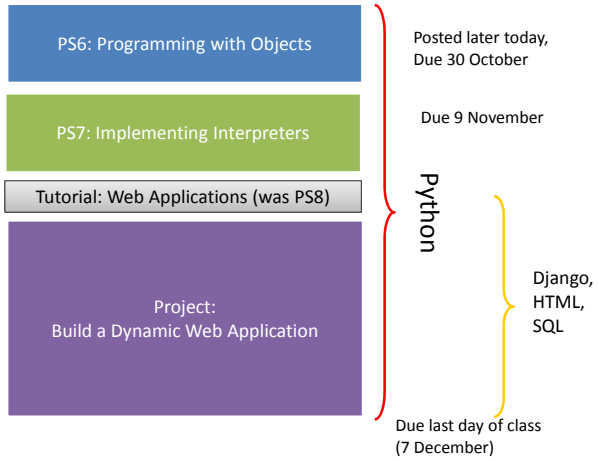
David Evans

Menu

- PS6, PS7, Project
- Objects
- PS5 vs. "Real" Databases

Sorry, no Office Hours today! Email me to arrange another time.
I will have my usual office hours tomorrow morning (10:30-11:30).

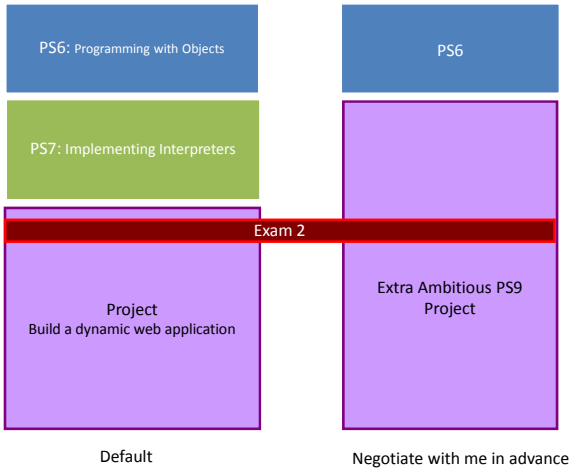
Remaining Problem Sets



Project Assignment

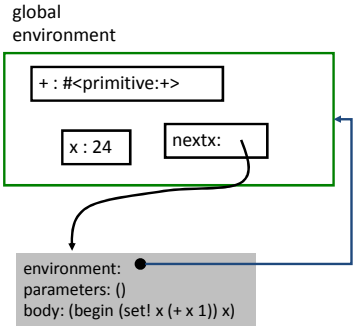
Problem: Make an interesting dynamic web site.

- Teams of 1-61 students
  - Mostly teams of 3 (need a good reason for a smaller or larger team)
  - You can form your own teams, or be assigned team
  - You will be asked to rate your teammates at the end of the project
- Can be anything you want that:
  - Involves interesting computation
  - Follows University's use policies (or on external server)
  - Complies with ADA Section 508 (accessible)



from Class 22: nextx

```
(define x 0)
(define (nextx)
  (set! x (+ x 1))
  x)
> (nextx)
1
> (set! x 23)
> (next x)
24
```



A Better Counter

- The place that keeps track of the count should be part of the counter, not part of the global environment
  - Can have more than one counter
  - Counter state is *encapsulated*: can only be modified by counter procedure
- Can we do this?

Stateful Application Rule:

To apply a constructed procedure:

1. **Construct a new environment**, whose parent is the environment of the applied procedure.
2. **For each procedure parameter, create a place** in the frame of the new environment with the name of the parameter. Evaluate each operand expression in the environment or the application and initialize the value in each place to the value of the corresponding operand expression.
3. **Evaluate** the body of the procedure **in the newly created environment.** The resulting value is the value of the application.

A Better Counter

```
(define (make-counter)
  ((lambda (count)
     (lambda ()
       (set! count (+ 1 count))
       count))
   0))
```

Sweeter Version

```
(define (make-counter)
  (let ((count 0))
    (lambda ()
      (set! count (+ 1 count))
      count)))
```

This is easier to read (syntactic sugar), but means the same thing. The place for count is created because of the application that is part of the let expression.

(**let** ((Name$_1$ Expression$_1$) (Name$_2$ Expression$_2$) ... (Name$_k$ Expression$_k$)) Expression$_{body}$)

is equivalent to

((**lambda** (Name$_1$ Name$_2$ . . . Name$_k$) Expression$_{body}$) Expression$_1$ Expression$_2$ . . . Expression$_k$)

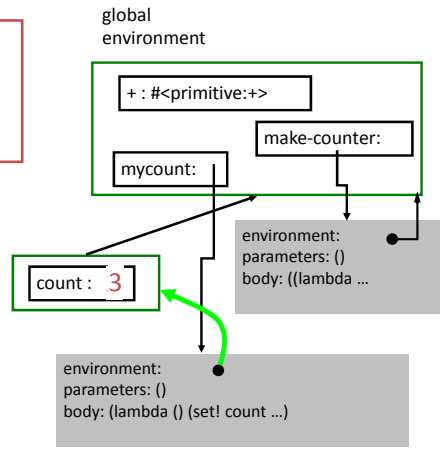
```
(define (make-counter)
  (let ((count 0))
    (lambda ()
      (set! count (+ 1 count))
      count)))
```

Draw the environment after evaluating:

```
> (define mycount
    (make-counter))
> (mycount)
1
> (mycount)
2
```

```
(define (make-counter)
  ((lambda (count)
     (lambda ()
       (set! count (+ 1 count))
       count))
   0))
```

```
> (define mycount
    (make-counter))
> (mycount)
1
> (mycount)
2
> (mycount)
3
```



Versatile Counter

```
(define (make-counter)
  ((lambda (count)
     (lambda ()
       (set! count (+ 1 count))
       count))
   0))
```

How can we make a counter that can do things other than just add 1?

An Even Sweeter Counter

```
(define (make-counter)
  (let ((count 0))
    (lambda (message)
      (cond ((eq? message 'reset!) (set! count 0))
            ((eq? message 'next!)
               (set! count (+ 1 count)))
            ((eq? message 'current) count)
            (else
             (error "Unrecognized message"))))))
```

Using Counter

```
> (define bcounter (make-counter))
> (bcounter 'next)
> (bcounter 'next)
> (bcounter 'next)
> (bcounter 'how-many)
3
> (bcounter 'reset)
> (bcounter 'how-many)
0
```

Objects

An *object* packages:

- **state** ("instance variables")
- **procedures** for manipulating and observing that state ("methods")

Why is this useful?

Problem-Solving Strategies

- PS1-PS4: Functional Programming
  - Focused on **procedures**
  - Break a problem into procedures that can be composed
- PS5: Imperative Programming
  - Focused on **data**
  - Design data for representing a problem and procedures for updating that data
- PS6: "Object-Oriented Programming"
  - Focused on objects that package state and procedures
  - Solve problem by designing objects that model the problem
  - Lots of problems in real (and imaginary) worlds can be thought of this way

PS5

How are commercial databases different from what you implemented for PS5?

UVa's Integrated Systems Project to convert all University information systems to use an Oracle database was originally budgeted for **$58.2 Million** (starting in 1999). Actual cost ended up over $100 Million.

http://www.virginia.edu/isp/

www.virginia.edu/isp/timeline.html

Real Databases

Atomic Transactions

a transaction may involve many modifications to database tables, but the changes should only happen if the whole transaction happens (e.g., don't charge the credit card unless the order is sent to the shipping dept)

Security

limit read/write access to tables, entries and fields

Storage

efficiently store data on disk, backup mechanisms

Scale

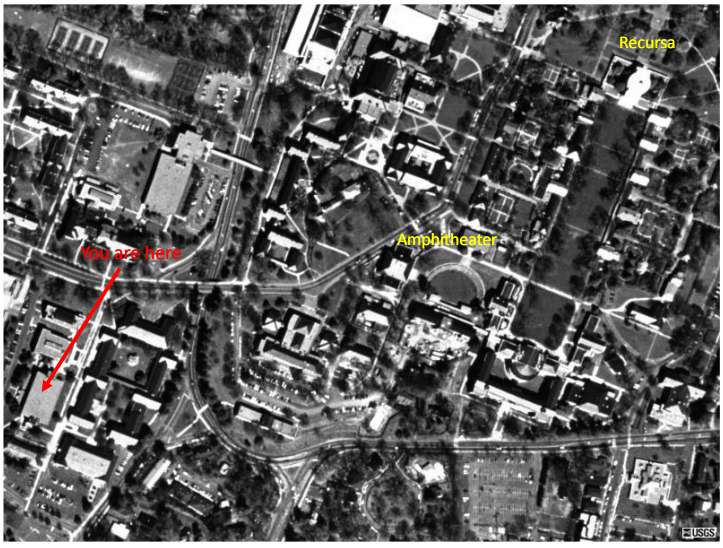
support really big data tables efficiently

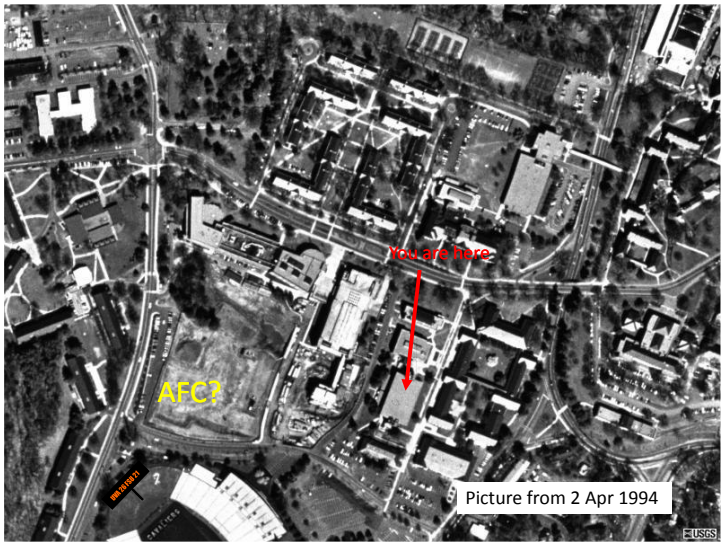
How big are big databases?

Microsoft TerraServer

Claimed biggest in **1998**

Aerial photos of entire US (1 meter resolution)





Big Databases Today

- Microsoft TerraServer
  - 3.3 Terabytes (claimed biggest in 1998)
- Internal Revenue Service
  - 150 Terabytes
- Wal-Mart
  - > 500 Terabytes (2003)
- Yahoo! (2008)
  - **2 Petabytes**
  - Analyze behavior of 500 M web visitors per month
- National Energy Research Scientific Computing
  - **2.6 Petabytes**
  - Atomic energy research, high-energy physics
  - Each particle collision generate > 30 KB

1 Petabyte = 1000 Terabytes = 10^{15} bytes

Lots more information to be collected: telephone calls in one year ~ 20 Exabytes

1 Exabyte = 1000 Petabytes = 10^{18} bytes

How much work?

table-select is in $\Theta(n)$ where n is the number of entries in the table

Would your table-select work for Wal-Mart?

If 1M entry table takes 1s, how long would it take Wal-Mart to select from >500TB ~ 2 Trillion Entries?

2 000 000s ~ 23 days

How do expensive databases perform table-select so much faster?

Indexing is the key! See Section 8.2.3

Making Objects in Scheme

```
(define (make-counter)
  (let ((count 0))
    (lambda (message)
      (cond ((eq? message 'reset!) (set! count 0))
            ((eq? message 'next!)
               (set! count (+ 1 count)))
            ((eq? message 'current) count)
            (else
             (error "Unrecognized message"))))))
```

Python Version

```
class Counter:
  def __init__(self):
    self.count = 0
  def reset(self):
    self.count = 0
  def current(self):
    return self.count
  def advance(self):
    self.count = self.count + 1
```

class defines a new class

The **__init__** method is special: it constructs a new object of the class.

self is the object we are creating (for __init__) or manipulating (for the other methods).

self.count = 0 (like (let ((count 0)) …)
In __init__: creates a new place named count as part of this object's frame, and initializes its value to **0**.

Python's built-in support for objects should (soon) make this easier to read and understand than the Scheme object system.

Charge

- PS6 will be posted tonight
- Wednesday: Python, Object-Oriented Programming
- Friday: "Golden Age of Science"

Start thinking about Project ideas

If you want to do an "extra ambitious" project (instead of PS7) convince me your idea is worthy before November 1