Welcome to 61A Lab!

We will begin at **5:10**!

Slides: **cs61a.bencuan.me**

Announcements

- Scheme
  - Part 1 due tonight
  - Parts 2-3 due next Tues.
  - Part 4 due 4/26

The Plan

- Data Abstraction

Data Abstraction

What is abstraction?

- Probably the single most important topic in 61A
  - Necessity for working with large codebases (61B, 162, your job…)
- Definition:

4. the process of considering something independently of its associations, attributes, or concrete accompaniments.

in other words: **use functions without caring about how they're implemented**

- You've done this lots and lots of times already

Example

```
def Cat(name, age):
    return [name, age]

def name(cat):
    return cat[0]

def age(cat):
    return cat[1]

===

def birthday(cat):
    print('Happy birthday', cat[0])
```

Example

```
def Cat(name, age):
    return [age, name]

def name(cat):
    return cat[1]

def age(cat):
    return cat[0]

===

def birthday(cat):
    print('Happy birthday', cat[0])
```

Example

```
def Cat(name, age):
    return [name, age]

def name(cat):
    return cat[0]

def age(cat):
    return cat[1]

===

def birthday(cat):
    print('Happy birthday', name(cat))
```

Abstraction Barrier

Public access

Cat()

age()

name()

Implementation only (don’t use!)

cat = [name, age]

cat[0]

cat[1]

Constructors vs Selectors

Constructors let users create new objects

Cat()

Selectors let users access data from those objects

name(cat)

age(cat)

Now, in scheme!

```
(define (cat name age)
    (cons name (cons age nil))

(define (name cat)
    (car cat)
)

(define (age cat)
    (car (cdr cat))
)

===

(define (birthday cat)
    (print ‘happy_birthday)
    (print (name cat))
)
```

Lab Hints

- **Don’t violate the abstraction barrier!!** Only use selectors when implementing functions
  - You may access the underlying data structure inside the selectors themselves
- Think about OOP design
  - OOP is just a fancy rule set for enabling data abstraction

Work Time!

go.cs61a.org/ben-queue