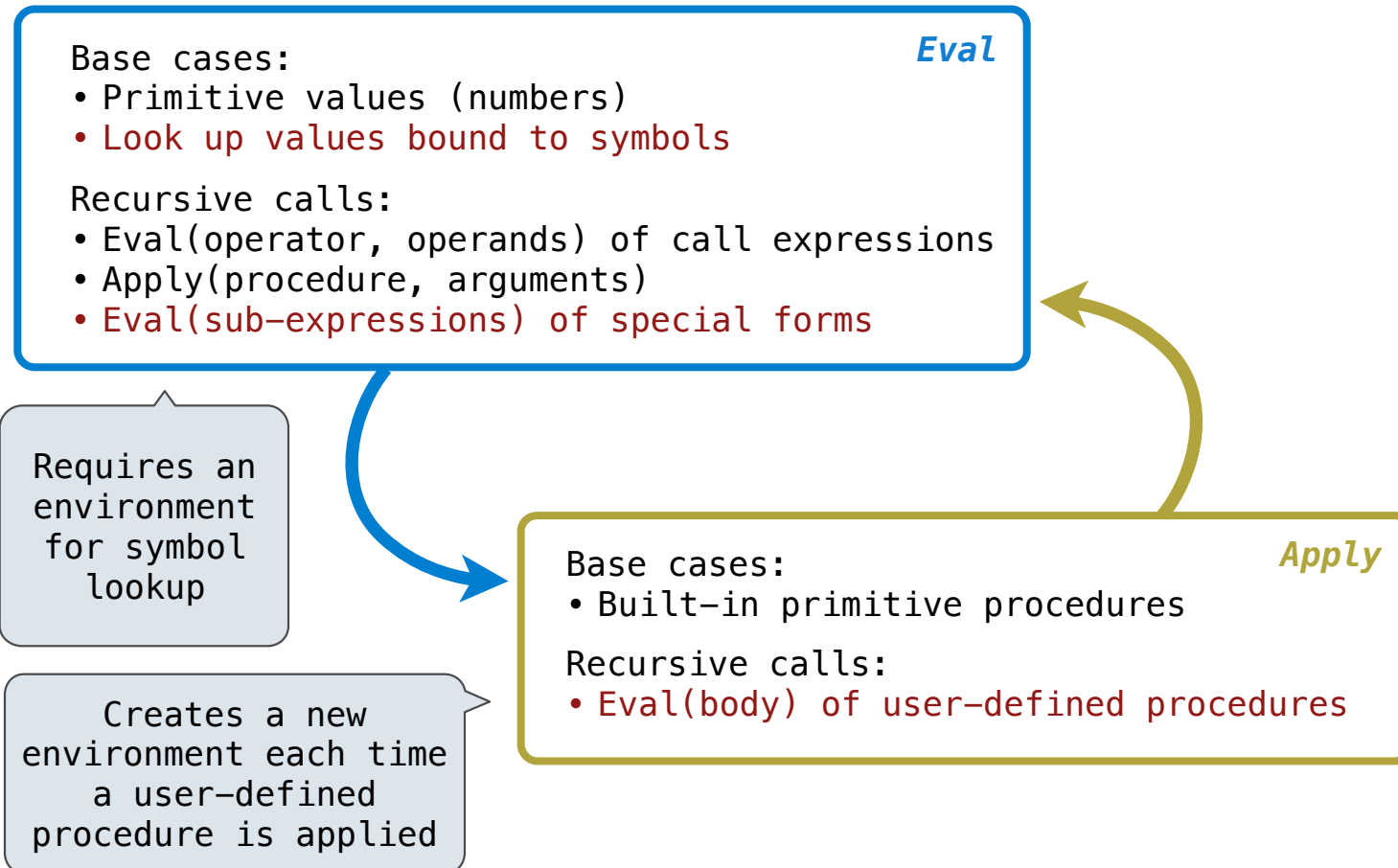# 61A Lecture 27

Wednesday, November 5

Announcements

- Homework 7 due Wednesday 4/8 @ 11:59pm
  - Homework party Tuesday 4/7 5pm–6:30pm in 2050 VLSB
- Project 1, 2, & 3 composition revisions due Monday 4/13 @ 11:59pm
- Quiz 2 released Tuesday 4/7 & due Thursday 4/9 @ 11:59pm
  - Open note, open interpreter, closed classmates, closed Internet
- Project 4 due Thursday 4/23 @ 11:59pm (Big!)

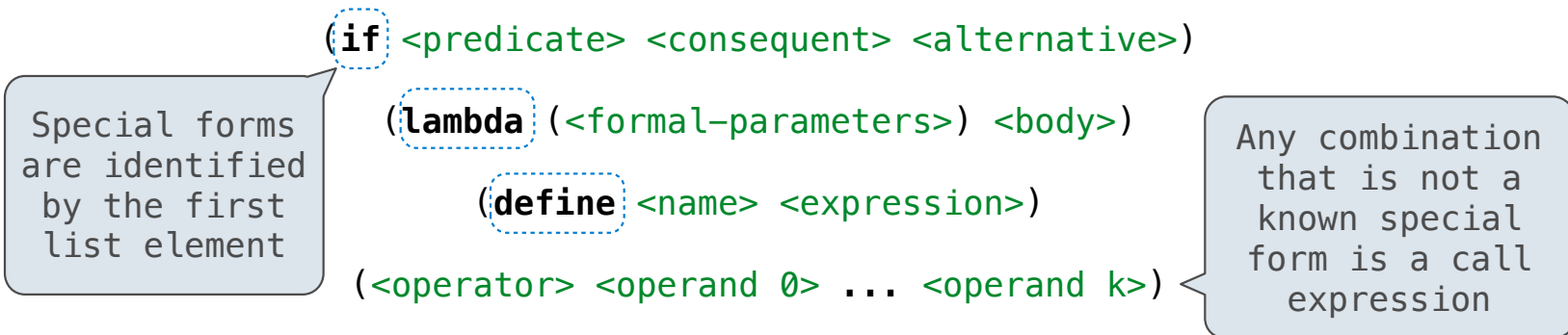# Interpreting Scheme

The Structure of an Interpreter

Base cases:
- Primitive values (numbers)
- Look up values bound to symbols

Recursive calls:
- Eval(operator, operands) of call expressions
- Apply(procedure, arguments)
- Eval(sub-expressions) of special forms

Eval

Requires an environment for symbol lookup

Base cases:
- Built-in primitive procedures

Recursive calls:
- Eval(body) of user-defined procedures

Apply

Creates a new environment each time a user-defined procedure is applied

Special Forms

Scheme Evaluation

The scheme_eval function choose behavior based on expression form:

• Symbols are looked up in the current environment

• Self-evaluating expressions are returned as values

• All other legal expressions are represented as Scheme lists, called combinations

(**if** <predicate> <consequent> <alternative>)

> Special forms
> are identified
> by the first
> list element

(**lambda** (<formal-parameters>) <body>)

(**define** <name> <expression>)

> Any combination
> that is not a
> known special
> form is a call
> expression

(<operator> <operand 0> ... <operand k>)

(define (demo s) (if (null? s) '(3) (cons (car s) (demo (cdr s)))))

(demo (list 1 2))

Logical Forms

Logical Special Forms

Logical forms may only evaluate some sub-expressions

- **If** expression: (**if** <predicate> <consequent> <alternative>)

- **And** and **or**:   (**and** <e1> ... <en>),   (**or** <e1> ... <en>)

- **Cond** expression: (**cond** (<p1> <e1>) ... (<pn> <en>) (else <e>))

  The value of an if expression is the value of a sub-expression:

  - Evaluate the predicate.   do_if_form

  - Choose a sub-expression: <consequent> or <alternative>.

  - Evaluate that sub-expression in place of the whole expression.

                                                        scheme_eval

                              (Demo)
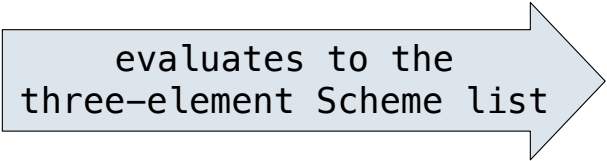
Quotation

Quotation

The quote special form evaluates to the quoted expression, which is not evaluated

(quote <expression>)   (quote (+ 1 2))   evaluates to the three—element Scheme list ⟹   (+ 1 2)

The <expression> itself is the value of the whole quote expression

'<expression> is shorthand for (quote <expression>)

(quote (1 2))   is equivalent to   '(1 2)

The scheme_read parser converts shorthand ' to a combination that starts with quote

(Demo)

Lambda Expressions

Lambda Expressions

Lambda expressions evaluate to user-defined procedures

      (**lambda** (<formal-parameters>) <body>)

      (**lambda** (x) (* x x))

```python
class LambdaProcedure:
    def __init__(self, formals, body, env):
        self.formals = formals .............. A scheme list of symbols
        self.body = body .............. A scheme expression
        self.env = env .............. A Frame instance
```

Frames and Environments

A frame represents an environment by having a parent frame

Frames are Python instances with methods **lookup** and **define**

In Project 4, Frames do not hold return values

```
g: Global frame
          y  |  3
          z  |  5

f1: [parent=g]
          x  |  2
          z  |  4
```

(Demo)

Define Expressions

Define Expressions

Define binds a symbol to a value in the first frame of the current environment.

  (**define** <name> <expression>)

1. Evaluate the <expression>

2. Bind <name> to its value in the current frame

  (**define** x (+ 1 2))

Procedure definition is shorthand of define with a lambda expression

  (**define** (<name> <formal parameters>) <body>)
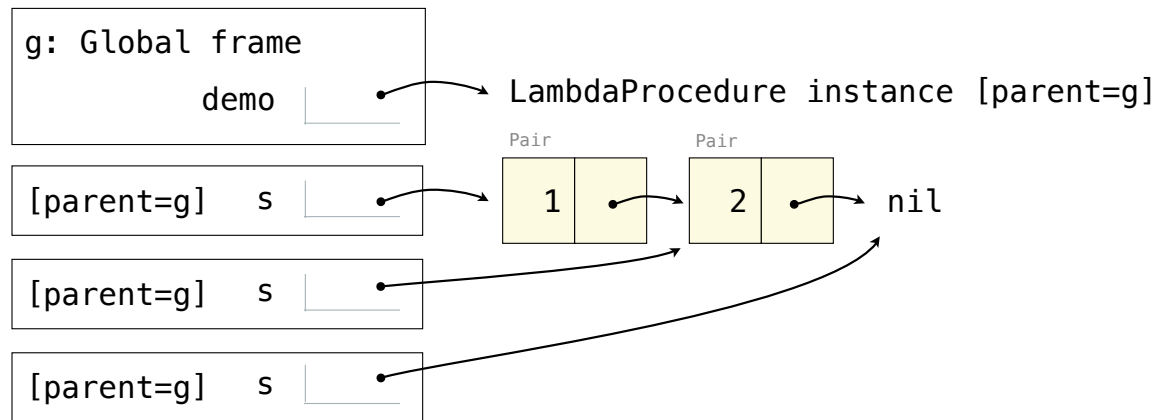
  (**define** <name> (**lambda** (<formal parameters>) <body>))

Applying User-Defined Procedures

To apply a user-defined procedure, create a new frame in which formal parameters are bound to argument values, whose parent is the **env** attribute of the procedure

Evaluate the body of the procedure in the environment that starts with this new frame

```
(define (demo s) (if (null? s) '(3) (cons (car s) (demo (cdr s)))))
```

```
(demo (list 1 2))
```

Eval/Apply in Lisp 1.5

$apply[fn;x;a] =$

$[atom[fn] \rightarrow [eq[fn;CAR] \rightarrow caar[x];$

$eq[fn;CDR] \rightarrow cdar[x];$

$eq[fn;CONS] \rightarrow cons[car[x];cadr[x]];$

$eq[fn;ATOM] \rightarrow atom[car[x]];$

$eq[fn;EQ] \rightarrow eq[car[x];cadr[x]];$

$T \rightarrow apply[eval[fn;a];x;a]];$

$eq[car[fn];LAMBDA] \rightarrow eval[caddr[fn];pairlis[cadr[fn];x;a]];$

$eq[car[fn];LABEL] \rightarrow apply[caddr[fn];x;cons[cons[cadr[fn];$

$caddr[fn]];a]]]$

$eval[e;a] = [atom[e] \rightarrow cdr[assoc[e;a]];$

$atom[car[e]] \rightarrow$

$[eq[car[e],QUOTE] \rightarrow cadr[e];$

$eq[car[e];COND] \rightarrow evcon[cdr[e];a];$

$T \rightarrow apply[car[e];evlis[cdr[e];a];a]];$

$T \rightarrow apply[car[e];evlis[cdr[e];a];a]]$