Announcements

- Project 1 composition revisions due Thursday 11/7 @ 11:59pm.
- Homework 8 due Tuesday 11/12 @ 11:59pm, and it’s in Scheme!
- Project 4 due Thursday 11/21 @ 11:59pm, and it’s a Scheme interpreter!
- New Policy: An improved final exam score can make up for low midterm scores.
  - If you scored less than 60/100 midterm points total, then you can earn some points back.
  - You don’t need a perfect score on the final to do so.

The Structure of an Interpreter

Interpreting Scheme

Scheme Evaluation

Special Forms

```
(define (demo s) (if (null? s) '(3) (cons (car s) (demo (cdr s)))))
(demo (list 1 2))
```
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 exp’**: (cond (<p1> <e1>) ... (<pn> <en>) (else <e>))

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

- Evaluate the predicate.
- Choose a sub-expression: <consequent> or <alternative>.
- Evaluate that sub-expression in place of the whole expression.

Quotation

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

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

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

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

The scheme_read parser converts shorthand to a combination.

Lambda Expressions

Lambda expressions evaluate to user-defined procedures.

(lamb (<formal-parameters>) <body>)

(lamb (x) (* x x))
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.

<table>
<thead>
<tr>
<th>Global frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>y 3</td>
</tr>
<tr>
<td>z 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>f1: (parent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x 2</td>
</tr>
<tr>
<td>z 4</td>
</tr>
</tbody>
</table>

---

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> (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 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)))))
```

Dynamic Scope

```
apply[fn,x,a] = [atom[n] = eq[n,CAR] = car[x];
                eq[n,CDR] = cdr[x];
                eq[n,CONS] = cons[car[n]];cdr[car[n]]];
                eq[n,ATOM] = atom[car[n]];eq[n,BQ] = eq[car[n];cadr[x]];
                T = apply[eval[fn][a]];]
        eq[car[n],LAMBDA] = eval[addrr[n];pair[cons[cons[addrr[n]];cadr[fn];]]];
        eq[car[n],LABEL] = apply[addrr[n];cons[cons[addrr[n];cadr[fn];]]];
        apply[fn][x][a] = [atom[x] = cadr[assoc[fn]];atom[car[x]] =
        [eq[car[x],QUOTE] = car[x];eq[car[x],COND] = eqn[cons[addrr[x];a]];
        T = apply[car[x];ev[n][cdr[x];a]]; T = apply[car[x];ev[n][cdr[x];a]];]
```
Dynamic Scope

The way in which names are looked up in Scheme and Python is called **lexical scope** (or **static scope**).

**Lexical scope**: The parent of a frame is the environment in which a procedure was **defined**.

**Dynamic scope**: The parent of a frame is the environment in which a procedure was **called**.

\[
\begin{align*}
\text{define } f & \text{ (lambda (x) (+ x y))} \\
\text{define } g & \text{ (lambda (x y) (f (+ x x)))} \\
(g 3 7)
\end{align*}
\]

**Lexical scope**: The parent for \(f\)'s frame is the global frame.

**Dynamic scope**: The parent for \(f\)'s frame is \(g\)'s frame.

**Error**: unknown identifier: \(y\)