Scheme Evaluation

The scheme_eval function dispatches on expression form:

- Symbols are bound to values in the current environment.
- Self-evaluating primitives are called atoms in Scheme.
- All other legal expressions are represented as Scheme lists.

\[
\text{if} \ <\text{predicate}> \ <\text{consequent}> \ <\text{alternative}>
\]

Logical Special Forms

Logical forms may only evaluate some sub-expressions.

- If expression: \((\text{if} \ <\text{predicate}> \ <\text{consequent}> \ <\text{alternative}>\)
- And and or: \((\text{and} <\text{e1}> ... <\text{en}>), \ (\text{or} <\text{e1}> ... <\text{en}>)\)
- Cond expr’n: \((\text{cond} (<p1> \ <e1>)) ... (<p_n> \ <e_n>) \ (\text{else} \ <e>)\)

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

\[
\text{do_if_form}
\]

Quotation

The quote special form evaluates to the quoted expression.

\[
\text{quote} \ <\text{expression}>
\]

Evaluates to the \(<\text{expression}>\) itself, not its value!

\text{quote} \ (<\text{expression}>)

\text{quote} \ (<\text{expression}>)

Lambda Expressions

Lambda expressions evaluate to user-defined procedures.

\[
\text{lambda} \ (<\text{formal-parameters}> \ <\text{body}>)
\]

\[
\text{lambda} \ (<x> \ (* \ x \ x))
\]

class LambdaProcedure(object):
    def __init__(self, formals, body, env):
        self.formals = formals
        self.body = body
        self.env = env

        self.formals = formals
        self.body = body
        self.env = env

        \[
        \text{scheme_eval}
        \]

        \[
        \text{do_if_form}
        \]

        \[
        \text{quote} \ (<\text{expression}>)
        \]

        \[
        \text{quote} \ (<\text{expression}>)
        \]

Demos

Example code snippets and explanations are provided for each section.
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>Frames and Environments</th>
</tr>
</thead>
<tbody>
<tr>
<td>A frame represents an environment by having a parent frame. Frames are Python instances with methods <code>lookup</code> and <code>define</code>. In Project 4, Frames do not hold return values.</td>
</tr>
</tbody>
</table>

![Frame Diagram]

Define Expressions

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

```
(define <name> <expression>)
```

Evaluate the `<expression>`.

Bind `<name>` to the result (define method of the current frame).

```
(define x 2)
```

Procedure definition is a combination of `define` and `lambda`.

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

Apply User-Defined Procedures

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 (f s) (if (null? s) '(3) (cons (car s) (f (cdr s)))))
(f (list 1 2))
```

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.

```
(define f (lambda (x) (+ x y)))
(define g (lambda (x y) (f (+ x x))))
(g 3 7)
```

**Lexical scope:** The parent for f's frame is the global frame. 
**Error:** unknown identifier: y

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