With Apologies to Robert Frost

A GOD’S LAMENT

Some said the world should be in Perl;
Some said in Lisp.
Now, having given both a whirl,
I held with those who favored Perl.
But I fear we passed to men
A disappointing founding myth,
And should we write it all again,
I’d end it with
A close-paren.

https://xkcd.com/312/
You now know most of CS17 Scheme

• Rules of evaluation:
  number rule, Boolean rule, symbol rule, list-eval rule, lambda rule, quote rule, cond rule, and rule, or rule.

  There is one more special form, if. This is discussed in homework. However, if is not vital because cond suffices.

There are still a couple things you don’t know about lambda expressions. Coming soon.

• Notion of binding symbols to data objects:
  Binding happens in two ways:
  • At the top level, using define. Adds a binding to the top-level environment.
  • Temporary bindings created when a procedure is applied to actual arguments.

There are also a few more built-in procedures you need to learn.
Quiz: Evaluate Scheme Expressions

Tell us the data object, if any, that results from evaluation.
(If not an expression or there is an error during evaluation, say why.)

1.  (1 + 2)
2.  evalpig
3.  (quote evalpig)
4.  (- (1 (- 1 1) 1))
5.  (- (- 1 1) 1)
6.  (+ pi (* 1 2))
7.  ((lambda (a) (* a (+ a 1))) 4)
8.  ((lambda (a) (* a b)) 3 4)
9.  ((lambda (a b) (or a b)) true false)
10. (cond (false 1) (true 2))
11. (cond ((= 2 (* 1 3)) (quote Sept))
    ((> 2 (* 1 3)) (1 + 2))
    (true (quote Oct)))
Imagine you write a procedure

```
(define quadratic-formula
  (lambda (a b c)
    (/ (+ (- b)(sqrt (- (* b b) (* 4 a c))))
       (* 2 a))))
```

and your friend wants to use it in their program. Suppose your friend has defined the variable \texttt{b}:

```
(define b (quote moo))
```

What happens when they try to apply your procedure?

```
(quadratic-formula 3 10 3)
```

When the procedure is applied to the actual arguments,
First step is to bind \texttt{a} to 3, \texttt{b} to 10, \texttt{c} to 3.
Will there be a conflict between the bindings of symbol \texttt{b}?

- Don’t want the procedure to do wrong thing.
- Also don’t want your friend’s code to mess up.

\textbf{Solution: Shadowing}
Bindings made in applying a procedure: *Shadowing*

```
(define quadratic-formula
  (lambda (a b c)
      ......))

(define b (quote moo))

(quadratic-formula 3 10 3)
```

**Design goals:**
- We don’t want the procedure to fail! In the procedure body, \( b \) should refer to the actual argument, 10.
- On the other hand, after the procedure application is over, we still want the variable \( b \) to be bound to the symbol `moo`.

**Solution:**
- New binding made in procedure application “hides” (shadows) top-level binding from the evalpig.
- Outside of procedure application, evalpig does not see the binding created in procedure application.
Environmental Studies

What if we apply `quadratic-formula` to some other numbers?

```
(quadratic-formula 1 5 1)
```

We don’t want the formal-to-actual bindings made during the previous application to matter in this application.

In previous lecture, I said that bindings made during procedure application are **erased** after the procedure application.

**Better way of thinking about it:**

- Each time a procedure is applied, a new set of bindings are created for the formal arguments.
- Those bindings form the context for **that** particular application of the procedure.
Each application of a procedure gets its own formal arguments.

We can take advantage of that: we can use one application to help another.
That is, the body of the procedure includes an application of the same procedure.

This is called recursion. This is the most important concept in CS17.
Overhanging bricks
Overhanging bricks

Measured in units of half-brick length, the overhang with 6 bricks is

\[ \frac{1}{6} + \frac{1}{5} + \frac{1}{4} + \frac{1}{3} + \frac{1}{2} + \frac{1}{1} \]

The overhang with \( n \) bricks is

\[ \frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \cdots + \frac{1}{3} + \frac{1}{2} + \frac{1}{1} \]

Called the \( n^{\text{th}} \) harmonic number, written \( H(n) \).

We want to compute how many bricks are needed to get an overhang of four brick-lengths.

Use the equation

\[ H(n) = \frac{1}{n} + H(n - 1) \]
\[ H(n) = \frac{1}{n} + H(n - 1) \]

How do you compute this in CS17 Scheme?
Simple in Scheme:

```
(define H
  (lambda (n)
    (+ (/ 1 n) (H (- n 1))))))
```

Okay, that’s too simple. It won’t ever finish.

```
(define H
  (lambda (n)
    (cond
      ((zero? n) 0)
      (#true (+ (/ 1 n) (H (- n 1))))))
```
Why does recursion work?
There’s no magic. Just apply the rules in the usual way.

But what is required for it to work?
For example, this won’t work:

```
(define H
  (lambda (n)
    (if (zero? n)
      0
      (+ (/ 1 n) (H (- n 1))))))
```

Each time procedure is applied, we call the event an *invocation*.

Why does recursion work?
There’s no magic. Just apply the rules in the usual way.

But what is required for it to work?  
For example, this won’t work:

```
(define H
  (lambda (n)
    (H n)))
```

**Key to recursion:** Recursive invocations must be addressing “easier” questions than original.

How to measure how “easy” a question is?
It depends.
Hack@Brown 2019

Applications

bit.ly/hab2019