Five useful questions (for any language)

1. What is the abstract syntax?
2. What are the values?
3. What environments are there?
4. How are terms evaluated?
5. What’s in the initial basis?
Scheme values: S-expressions

An S-expression is one of

- a symbol: 'Halligan 'tufts
- a literal integer: 0 77
- a literal Boolean: #t #f
- the empty list: '()
- (cons v₁ v₂), where v₁ and v₂ are S-expressions
S-Expression operators

Like any abstract data type, S-Expressions have:

- **creators** (create new values): ' ()
- **producers** (build new from existing):
  (cons s1 s2)
- **mutators** (change values; not found in μScheme)
- **observers** (examine values):
  number? symbol? boolean? null? pair?
  car cdr
Symbolic data: lists

Many predefined functions expect a list of S-expressions

A list of S-expressions is either
- the empty list ' ()
- \((\text{cons } v_1 v_2)\), where \(v_1\) is an S-expression and \(v_2\) is a list of S-expressions

We say “an S-expression followed by a list of S-expressions”
Lists defined inductively

\( \text{LIST}(A) \) is the smallest set satisfying this equation:

\[
\text{LIST}(A) = \{ '() \} \cup \{ (\text{cons} \ a \ as) \mid a \in A, as \in \text{LIST}(A) \}
\]

Equivalently, \( \text{LIST}(A) \) is defined by these rules:

\( (\text{EMPTY}) \)

\[
'() \in \text{List}(A)
\]

\( (\text{CONS}) \)

\[
a \in A \quad as \in \text{List}(A) \\
(\text{cons} \ a \ as) \in \text{List}(A)
\]
Equations and function for append

(append '() ys) == ys

(append (cons z zs) ys) == (cons z (append zs ys))

(define append (xs ys)

  (if (null? xs)

    ys

    (cons (car xs) (append (cdr xs) ys))))