Scheme implement 的思路概览

text: a string representation of an expression of a language.

eg. (+1 (-2 3))

$\xRightarrow{\text{lexical analysis}}$

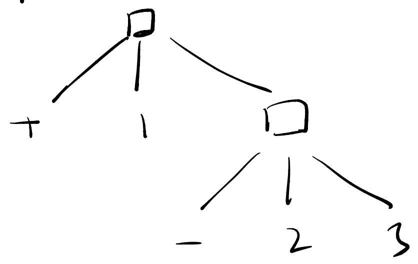
tokens:

eg. '(' '+' '1' '(' '-' '2' '3' ')' ')'.

$\Downarrow$ syntactic analysis

expression:

eg. expression tree



Pair(+, Pair(1, Pair(O, nil)))

↓

Pair(-, pair(2, Pair(3, nil)))

$\Leftarrow$

calc_eval & calc_apply: compute the value of an expression

else: ① interactive interpreters

② special forms, like <u>if, and, or, cond, lambda, define</u> statements.

lambda, solve with "class LambdaProcedure" (def __init__(self, formals, body, env):)

(define <name> <expression>) 1. evaluate the <expression>
2. Bind <name> to its value in the current frame.

(define (funcname para) <body>) $\Longleftrightarrow$ (define funcname (lambda (para) <body>))

③ frames and environments.

A frame represents an environment by having a parent frame.

Frames are Python instances with methods <u>lookup</u> and <u>define</u>.

④ Quotation: '(1 2) $\Longleftrightarrow$ (quote (1 2))

在 scheme_read 中, 自动将读取的 '(1 2) 转换为 (quote (1 2))

It should be the return value of the tail call is the return value of the current procedure call. Meaning, we can skip keeping around all the frames that we don't need because the return value for the last frame can return staight up to the original call. Tail calls shouldn't increase the environment size.

tail recursion