

## Scheme

"purely non-destructive operations"

Scheme 的数据类型可以分为 atoms 和 pairs.

### 1) Atoms

- Numbers: integer, floating-point, complex, rational
- Symbols 一种有点像字符串的数据类型.
- Booleans: #t, #f
- The empty list: ()
- Procedures(functions).

### 2) Pairs 类似 Python 中具有两个元素的元组.

pair 的组成部分既可以是 pair, 也可以是 atom. 可以用嵌套的方法构建更复杂的数据.

Pairs and Lists 注: 此处 List 的实现方法很像之前课程中的链表

缩写

含义

(V)	(V.())
(V <sub>1</sub> , V <sub>2</sub> , ..., V <sub>n</sub> )	(V <sub>1</sub> .(V <sub>2</sub> .(...(V <sub>n</sub> .())))))
(V <sub>1</sub> , V <sub>2</sub> , ..., V <sub>n-1</sub> , V <sub>n</sub> )	(V <sub>1</sub> .(V <sub>2</sub> .(...(V <sub>n-1</sub> , V <sub>n</sub> ))))      improper list 暂时不用掌握

一些可能的 Scheme 表达式如: (x=3) (+ (\* 3 7) (- x)) 等

Scheme 程序的奇妙之处在于 Scheme 程序看起来就像那些列表一样

"Scheme programs are Scheme data."

对于大多数 Scheme 表达式而言, 它们看起来就是表达式树.

• 对于 numerals, booleans, characters, strings 等表达式, 它们指代的就是自己.

我们称它们是 self-evaluating 的.

• 对于大部分的 lists (亦可称为 forms), 它们都 (近似地) 代表一个函数

(OP E<sub>1</sub>, ..., E<sub>n</sub>) 其中 OP 表示操作, E<sub>1</sub> 到 E<sub>n</sub> 为表达式.

(几乎) 等价于 Python 中的 OP(E<sub>1</sub>, ..., E<sub>n</sub>) 这种调用函数的表达式.

例如 (> 3 2)  $\Leftrightarrow$  3 > 2

(- (/ (\* (+ 3 7 10) (- 1000 8)) 992) 17)  $\Leftrightarrow$  ((3+7+10)\*(1000-8))/992-17

(pair? (list 1 2)) → #t

当我们需要将表达式而非它的值赋值给变量时，我们可以用 quote.

scm> (+ 1 2) | scm> (quote (+ 1 2)) | scm> '(+ 1 2) ⇒ quote 的缩写形式  
3 | (+ 1 2) | (+ 1 2)

### Special forms

实际上 (quote E) 就是一种 Special form. 执行该程序时不对 E 求值，而是返回它本身。

其他的 special forms 部分列举如下。

1) (quote E) 略

2) (if (> x y) x y) 类似 x if x > y else y

3) (and E<sub>1</sub> … E<sub>n</sub>) 类似 Python 中的 and

4) (or E<sub>1</sub> … E<sub>n</sub>) 类似 Python 中的 or

5) (lambda (x y) (/(\* x x) y)) 类似 Python 中的 lambda，得到一个函数。

6) (define pi 3.14) 定义变量，类似 Python 中的声明。

(define (f x) (\* x x)) 定义函数，类似 Python 中的 def 语句。

7) (cond ((< x 1) 'small) 美似于 Python 中的 if elif else 语句

((< x 3) 'medium)

(#t 'large))

### Pairs and Lists

用 cons 构建对和列表，用 car 和 cdr 取出其中的元素。

↓  
Link()  
↓  
L.first L.rest

当然也可以用 list 构建列表，使用表达式 (list E<sub>1</sub> … E<sub>n</sub>) 即可。

## Equivalence Operations

= only for numbers

equ? for numbers, empty list, symbols, ...

like Python's "is" elsewhere

eq? is Python's "is" (might not work for numbers)

equal? like "equ?" but also does deep equality for pairs and lists. 类似于 "=="

## Binding Constructs: Let

scm> (define x 1)

这是一个 derived form, 它等价于

scm> (let ((x 5) (y (+ x 2))) (+ x y))

scm> ((lambda (x y) (+ x y)) 5 (+ 5 2))

24

let 语句用于引入局部变量，其基本语法如下：

(let ( (var1 value1)

将 value1, 2 分别绑定至变量 var1, 2 上

      (var2 value2)

然后执行 body 部分的语句

...

)

body)