Lisp 入门

《计算的本质》系列课程

版权声明

联系方式

如果你想要购买本课程
请微信联系 xiedaimala02 或 xiedaimala03

如果你发现有人盗用本课程
请微信联系 xiedaimala02 或 xiedaimala03

Lisp 入门

- 学习目的
- 下载开发环境
- 基本语法
- 线性递归 v.s. 线性迭代

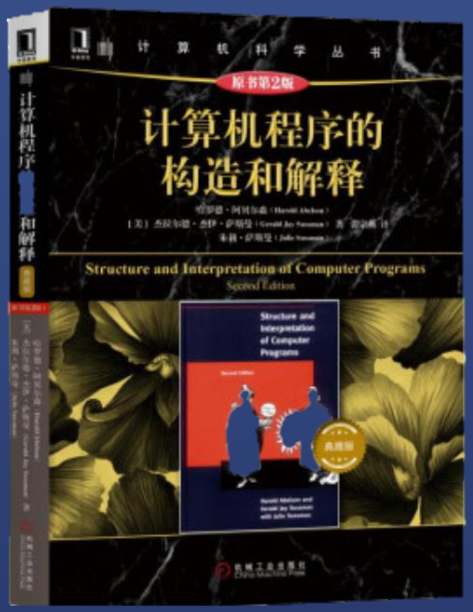
学习 Lisp 的目的

- 黑客与画家
- 目的
  - ✓ 就是为了了解一下远古程序员在想些什么
  - ✓ 顺便拓展一下视野
- 吐槽
  - ✓ 很多程序员大有「一辈子只学一门语言」的架势
  - ✓ 每当有多学一门语言的机会，他们总是会找借口逃避
  - ✓ 逃避可以让你舒服一阵子，但等你遇到瓶颈的时候...
  - ✓ 《逃避可耻，但有用》月薪娇妻

下载开发环境

- 运行 Scheme 的软件
  - Scheme 是 Lisp 的方言
  - 很多软件都可以运行 Scheme
  - 比如 MIT Scheme 和 Racket
  - 本节课以 Racket 为例
- 运行代码
  - 打开 Racket 直接输入代码
  - 打开 DrRacket 选择 Language 后点击 Run 再输入代码
  - 后者更友好一些，本质都一样

课外资料



- Structure and Interpretation of Computer Programs
  - ✓ 《计算机程序的构造和解释》，简称 SICP
  - ✓ 本节课大部分代码来自此书
  - ✓ 没有兴趣不要购买，大部分人看不懂

课外资料

- SICP 英文版
  - https://mitpress.mit.edu/sites/default/files/sicp/full-text/book/book-Z-H-4.html#%_toc_start
- SICP Python 描述中文版
  - https://wizardforcel.gitbooks.io/sicp-py/content/
- 对应的视频
  - https://www.bilibili.com/video/av8515129/

Lisp 基本语法

- 数字

```
486
```

- 四则运算

```
(+ 123 321)
(- 1000 334)
(* 5 99)
(/ 10 5)
```

- 小数

```
(+ 2.7 10)
```

- 注释

```
✓ 486 ;number
```

- 四则运算升级

```
(+ 21 35 12 7)
(* 25 4 12)
```

- 复杂一点

```
(+ (* 3 (+ (* 2 4) (+ 3 5))) (+ (- 10 7) 6))
```

等价于

```
(+ (* 3
      (+ (* 2 4)
         (+ 3 5)))
   (+ (- 10 7)
      6))
```

Lisp 基本语法 2

- 命名

```
> (define size 2)
> size
2
> (* 5 size)
10
```

这不就是变量吗?

非也! size 是不可变的!

所以不是变量，最多是常量

组合式的求值过程

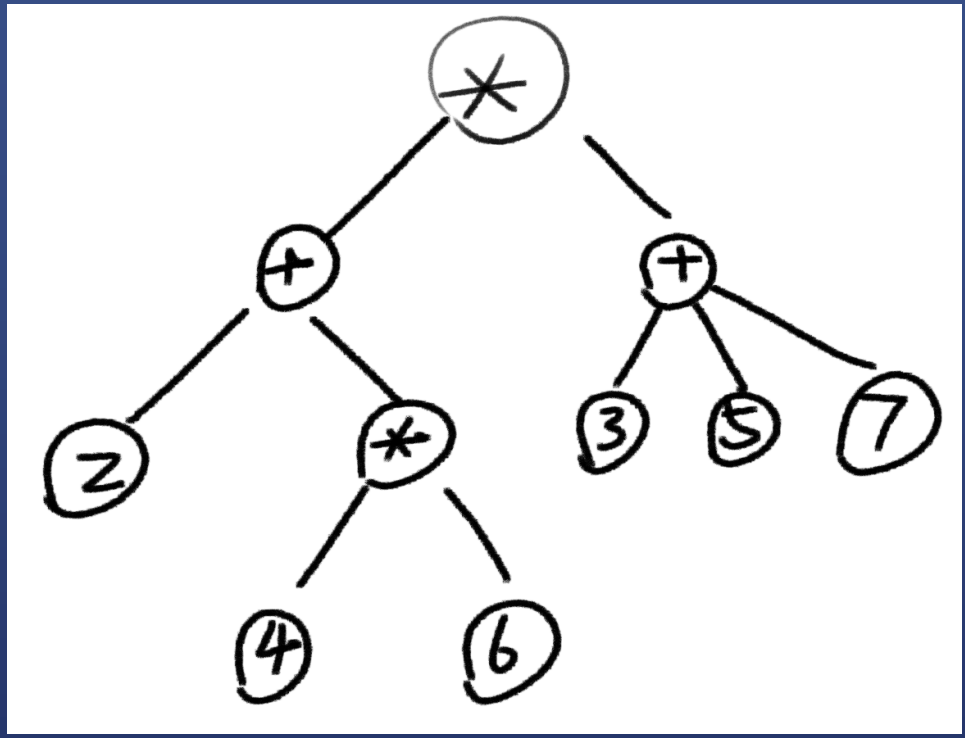
- 举例

```
(* (+ 2 (* 4 6))
   (+ 3 5 7))
```

要想求此表达式的值

- ✓ 求先各子表达式的值
- ✓ 然后把所有值作用于 *

这是一个递归的过程



Lisp 基本语法 3

- 复合过程（很像函数）

```
(define (square x) (* x x))
        函数名 形参   函数体
```

- ✓ 换句话说，当我们写 (square 6) 相当于写 (* 6 6)
- ✓ 相对地，+ 和 * 属于基本过程（内置函数）

- 举例

- ✓ 求 x^2 + y^2

```
(+ (square x) (square y))
(+ (* x x)    (* y y)   )
```

- ✓ 将这个过程定义成复合过程

```
(define (sum-of-squares x y)
  (+ (square x) (square y) ) )
(sum-of-squares 3 4)
> 25
```

代入法

- 代入求值

```
(define (f a)
  (sum-of-squares (+ a 1) (* a 2)))
```

- ✓ 求 (f 5)
- ✓ 代入即可（视频演示）

- 注意
- ✓ 代入法是为了教学，并不是解释器的实际工作方式

- 其实，代入法分两种

应用序 v.s. 正常序

- 代入的时候可以选择
  - 应用序：参数先求值，再代入过程（Scheme）
  - 正常序：参数不求值，全代入，最后再求值
  - 两种方式各有利弊，后面的课程可能涉及
- 后者举例

```
(f 5)
(sum-of-squares (+ 5 1) (* 5 2))
(+    (square (+ 5 1))    (square (* 5 2))  )
(+    (* (+ 5 1) (+ 5 1)) (* (* 5 2) (* 5 2)))
```

代入完毕，最后求值

```
(+          (* 6 6)              (* 10 10))
(+            36                    100)
136
```

Lisp 基本语法 4

- if...else... 怎么做?

```
(cond (<p1> <v1>)
      (<p2> <v2>)
      ...
      (<pn> <vn>))
```

- 举例：求 x 的绝对值

```
(define (abs x)
  (cond ( (> x 0) x)
        ( (= x 0) 0)
        ( (< x 0) (- x) )))
```

Lisp 基本语法 5

- if...else... 还能怎么做?

```
(if   <trueOrFalse>
      <e1>
      <e2>)
```

- 继续举例：求 x 的绝对值

```
(define (abs x)
  (if (< x 0) (- x) x) )
```

Lisp 基本语法 6

- 或且非

```
(and (> x 5) (< x 10))

(define (>= x y)
  (or (> x y) (= x y)))

(define (>= x y)
  (not (< x y)))
```

练习：牛顿法求平方根

- Scheme 代码

```
(define (sqrt-iter guess x)
  (if (good-enough? guess x)
      guess
      (sqrt-iter (improve guess x)
                 x)))
(define (improve guess x)
  (average guess (/ x guess)))
(define (average x y)
  (/ (+ x y) 2))
(define (good-enough? guess x)
  (< (abs (- (* guess guess) x)) 0.001))

(define (my_sqrt x)
  (sqrt-iter 1.0 x))
```

SICP 练习 1.6

- 我觉得是个很经典的题目

✓ 可以用 cond 实现 if 吗?

```
(define (new-if predicate then-clause else-clause)
  (cond (predicate then-clause)
        (else else-clause)))
```

✓ 可以用 new-if 代替 if 吗?

✓ 动手试试，会有惊喜!

语法小结

- 无括号
  - ✓ 就是个值
- 有括号
  - ✓ 可能是个过程（类似于函数）
  - ✓ 可能是个特殊操作（如 if）

内部定义

- 可以在过程里定义过程

```
(define (my_sqrt x)
  (define (good-enough? guess x)
    (< (abs (- (square guess) x)) 0.001))
  (define (improve guess x)
    (average guess (/ x guess)))
  (define (sqrt-iter guess x)
    (if (good-enough? guess x)
        guess
        (sqrt-iter (improve guess x) x)))
  (sqrt-iter 1.0 x))
```

- 此代码在 Racket 无法运行
  - ✓ 貌似是因为 Racket 做了限制，具体原因我不知道

词法作用域

- 没有必要把 x 传来传去

```
(define (my_sqrt x)
  (define (good-enough? guess)
    (< (abs (- (square guess) x)) 0.001))
  (define (improve guess)
    (average guess (/ x guess)))
  (define (sqrt-iter guess)
    (if (good-enough? guess)
        guess
        (sqrt-iter (improve guess))))
  (sqrt-iter 1.0))
```

- 根据词法可以确定的作用域
  - ✓ 函数可以访问其外部的自由变量（JS的闭包）

线性递归与线性迭代

你分得清吗?

阶乘 n! = n * n-1 * … * 1

- 线性递归形式

```
(define (f n)
  (if (= n 1)
      1
      (* n (f (- n 1)))))
```

- 使用代入法求 f(6)
  - ✓ 视频演示

阶乘

- 线性迭代形式

```
(define (f n)
  (iter 1 n 1))
(define (iter m end result)
  (if (> m end)
      result
      (iter (+ m 1) end (* m result))))
```

- 使用代入法求 f(6)

✓ 视频演示

✓ 发没发现，没有「递进」，没有「回归」

区别

- 线性递归

✓ 由于要回归，所以解释器必须在每次递进时记录回归时的信息，递进越深，要记住的回归信息越多

- 线性迭代

✓ 每次只记录三个变量，三个变量一直迭代

✓ 解释器只需要传递这三个变量到底即可

注意

- 线性递归和线性迭代都是「递归」
  - ✓ 线性递归和线性迭代强调的是「计算过程」
  - ✓ 「递归」表示函数调用自己这种「形式」

Lisp 有意思吗?

下节课再见