4. 定义函数

1.简介

在前面的章节中，我已经讲解了：

1.如何安装MIT-Scheme

2.Scheme解释器是如何对S-表达式求值

3.基本的表操作

在本章中，我会讲解如何自定义函数。正如Scheme是函数式编程语言，你需要通过编写小型函数来构造你的程序。因此，明白如何构造并组合这些函数对掌握Scheme尤为关键。

在前端定义函数非常不便，因此我们通常需要在编译器中编辑好代码，并在解释器中加载它们。

2.如何定义简单函数并加载它们

你可以使用define来将一个符号与一个值绑定。你可以通过这个操作符定义例如数、字符、表、函数等任何类型的参数【存疑】。

让我们使用任意一款编辑器（记事本亦可）来编辑代码片段1中展示的代码，并将它们存储为‘hello.scm’，放置在类似于‘C:\doc\scheme\’文件夹下。如果可以的话，把这些文件放在你在第一章定义的MIT-Scheme默认文件夹下。

代码片段1（hello.scm）

; Hello world as a variable

(define vhello "Hello world") ;1

; Hello world as a function

(define fhello (lambda () ;2

"Hello world"))

接下来，向Scheme解释器输入下面的命令：

(cd "C:\\doc\\scheme")

;Value 14: #[pathname 14 "c:\\doc\\scheme\\"]

(load "hello.scm")

;Loading "hello.scm" -- done

;Value: fhello

通过这些命令，hello.scm就被加载到解释器中。如果你的当前目录被设定在了脚本所在目录，那么你就不需要再输入第一行的命令了。然后，向解释器输入下面的命令：

vhello

;Value 15: "Hello world"

fhello

;Value 16: #[compound-procedure 16 fhello]

(fhello)

;Value 17: "Hello world"

操作符define用于申明变量并接受两个参数。Define运算符会使用第一个参数作为全局参数，并将其与第二个参数绑定起来。因此，代码片段第1行中，我们声明了一个全局参数vhello，并将其与“Hello，World”绑定起来。

紧接着，在第2行声明了一个返回“Hello World”的过程。

特殊形式lambda用于定义过程。Lambda需要至少一个的参数，第一个参数是由定义的过程所需的参数组成的表。因为本例fhello没有参数，所以参数表是空表。

在解释器中输入vhello，解释器返回“Hello，World”。如果你在解释器中输入fhello，它也会返回向下面这样的值：

**#[compound-procedure 16 fhello]**,

这说明了Scheme解释器把过程同常规数据类型用同样的方式对待。正如我们在前面章节讲解的那样，Scheme解释器通过内存空间中的数据地址操作所有的数据，因此，所有存在于内存空间中的对象都以同样的方式处理。

如果把fhello当过程对待，你应该用括号扩住这些符号，比如(fhello)。

然后解释器会按照第二章讲述的规则那样对它求值，并返回“Hello World”。

3 定义有参数的函数

可以通过在lambda后放一个参数表来定义有参数的函数。将代码片段2保存为farg.scm并放在同hello.scm一致的目录。

[code 2] (farg.scm)

; hello with name

(define hello

(lambda (name)

(string-append "Hello " name "!")))

; sum of three numbers

(define sum3

(lambda (a b c)

(+ a b c)))

保存文件，并在解释器中载入此文件，然后调用我们定义的函数。

(load "farg.scm")

;Loading "farg.scm" -- done

;Value: sum3

(hello "Lucy")

;Value 20: "Hello Lucy!"

(sum3 10 20 30)

;Value: 60

Hello

函数hello有一个参数(name)，并会把“Hello”、name的值、和“！”连结在一起并返回。

预定义函数string-append可以接受任意多个数的参数，并返回将这些参数连结在一起后的字符串。

Sum3

此函数有三个参数并返回这三个参数的和。

4. 一种函数定义的短形式

用lambda定义函数是一种规范的方法，但你也可以使用类似于代码片段3中展示的短形式。

代码片段3

; hello with name

(define (hello name)

(string-append "Hello " name "!"))

; sum of three numbers

(define (sum3 a b c)

(+ a b c))

在这种形式中，函数按照它们被调用的形式被定义。代码片段2和代码片段3都是相同的。有些人不喜欢这种短形式的函数定义，但是我在教程中使用这种形式，因为它可以使代码更短小。

练习1

按照下面的要求编写函数。这些都非常简单但实用。

1. 将参数加1的函数

2. 将参数减1的函数

练习2

让我们按照下面的步骤编写一个用于计算飞行距离的函数。

1. 编写一个将角的单位由度转换为弧度的函数。180度即π弧度。π可以通过下面的式子定义：

(define pi (\* 4 (atan 1.0)))

2. 编写一个用于计算按照一个常量速度（水平分速度）运动的物体，t秒内的位移的函数。

3. 编写一个用于计算物体落地前的飞行时间的函数，参数是垂直分速度。

忽略空气阻力并取重力加速度g为9.8m/s^2

提示：设落地时瞬时竖直分速度为 **-vy**，有如下关系：

2 vy = g t   
此处t为落地时的时间。

4. 使用问题1-3中定义的函数编写一个用于计算一个以初速度v和角度theta掷出的小球的飞行距离。

提示：首先，将角度的单位转换为弧度（假定转换后的角度为theta1）。初始水平、竖直分速度分别表示为：**v** cos(**theta1**) and **v** sin(**theta1**)。落地时间可以通过问题3中定义的函数计算。由于水平分速度不会改变， 因此可以利用问题2中的函数计算距离。

5.

5. 关于编辑器

这里，我会推荐一些能非常方便地编辑Scheme代码的编辑器。

5.1 Emacs

Emacs21的Windows版本可以从<http://ftp.gnu.org/gnu/emacs/windows/>找到，下载emacs-21.3-bin-i386.tar.gz并解压它。

你会在bin文件夹下发现一个叫runemacs.exe的可执行文件。双击这个程序来启动编辑器。尽管键位布局和Windows的标准相当不同，但是因为有一个菜单栏和鼠标控制器而显得相当用户友好。你也可以通过编辑名为.emacs的配置文件来实现自定义配置。编辑器提供了一种Scheme模式，此模式下能够编辑器能识别预定义单词，以及通过Ctri-i或TAB键来自动缩进。除此之外，当一个输入一个右括号后，编辑器会自动显示与之匹配的左括号。

在Windows系统中，emacs不能够与MIT-Scheme进行交互。你只能手动地储存并加载源代码。但从另一个方面来说，在UNIX和Linux系统下，emacs可以同MIT-Scheme进行交互式地调用，因此编辑代码也可以在交互中完成。

5.2 Edwin

Edwin是MIT-Scheme配备的编辑器。它有点像emacs18。但它没有菜单栏和鼠标控制，因此显得不太用户友好。只有少数人用这个编辑器，因此网络上可用的说明也很少。虽然如此，你可以使用这个编辑器进行交互式的代码编辑。我在Windows上使用这个编辑器编辑Scheme代码。

如何使用Edwin

双击Edwin的图标以启动Edwin。当Edwin启动后，一个叫\*Scheme\*的默认缓冲区出现在屏幕上，它对应于emacs中的\*scratch\*缓冲区。你可以将\*scheme\*用作解释器前端。按下Ctrl-X Ctrl-e 就可以对S-表达式进行求值。

1. 文件的打开与关闭，编辑器的关闭

按下Ctrl-X Ctrl-F来打开一个文件。如果你指定的文件并不存在，则会创建一个新文件。初始路径被设置为了‘C:\’，你在打开文件前应该修改这个路径。

按下Ctrl-X Ctrl-S来保存文件，而按下Ctrl-x Ctrl-w则是文件另存为。退出编辑器请按下Ctrl-x Ctrl-c。

1. 缩进

按下Ctrl-i或者TAB可以缩进。

1. 剪切，复制和粘贴

我们无法使用鼠标，因此复制（剪切）、粘贴起来就会显得不太方便。但你可以像下面这样做：

1. 首先，通过方向键将光标移动至待选区域的开头，然后按下Ctrl-SPACE。
2. 然后移动至结束位置按下Ctrl-w来剪切区域，按下Alt-w来复制区域。
3. 最后，移动至你想要复制的区域，按下Ctrl-y
4. 求值S-表达式

* 按键Alt-z用于求值以define开头的S-表达式。
* 按键Alt-:用于在一个小型的缓冲区中求值S-表达式。这个通常用在测试用Alt-z求值的函数。
* 按键Ctrl-x Ctrl-e用于求值整个\*scheme\*缓冲区中的S-表达式。

请查阅 <http://www-swiss.ai.mit.edu/projects/scheme/documentation/user_8.html> 以获得更多关于Edwin的帮助。你下载的MIT-Scheme中也附带了同样的文档。

6. 小结

本章红，我讲解了如何定义函数。特殊形式define用于定义函数和全局参数。我也讲解了用合适的编辑器（比如emacs）来编辑源代码，载入源码文件比在前端直接定义函数方便多了。

下个章节中，我讲介绍分支。

练习答案

答案1

; 1

(define (1+ x)

(+ x 1))

; 2

(define (1- x)

(- x 1))

答案2

代码如下所示：

; definition of pi

(define pi (\* 4 (atan 1.0)))

; degree -> radian

(define (radian deg)

(\* deg (/ pi 180.0)))

; free fall time

(define (ff-time vy)

(/ (\* 2.0 vy) 9.8))

; horizontal distance

(define (dx vx t)

(\* vx t))

; distance

(define (distance v ang)

(dx

(\* v (cos (radian ang))) ; vx

(ff-time (\* v (sin (radian ang)))))) ; t

向解释器中载入后，距离可以像这样计算：

(distance 40 30)

;Value: 141.39190265868385

函数返回一个合理的值：141.1米，因为忽略了空气阻力，所以这个值略微偏大。