**6. Local Variables**

**1.Introduction**

I have explained how to defined functions in the previous chapter. However, as the way how to define local variables has not been explained, the function definitions apt to be redundant, which makes writing complicated functions difficult.

In this chapter, I will explain about local variables, which makes defining functions easier.

**2. let expression**

Local variables can be defined using the **let** expression. The format is like as follows:

(**let** **binds** **body**)

Variables are declared and assigned to initial values in the **binds** form. The **body** consists of arbitrary numbers of S-expressions.  
The format of the **binds** form is like as follows:

[**binds**] → ((**p1** **v1**) (**p2** **v2**) ...)

Variables **p1**, **p2** ... are declared and bind them to the initial values, **v1**, **v2** ... The scope of variables is the **body**, which means that variables are valid only in the **body**.

Example 1: Declaring local variables **i** and **j**, binding them to 1 and 2 and then making a sum of them.

(let ((i 1) (j 2))

(+ i j))

;Value: 3

The let expressions can be nested.

Example 2: Making local variables **i** and **j**, binding them to 1 and **i**+2, and multiplying them.

(let ((i 1))

(let ((j (+ i 2)))

(\* i j)))

;Value: 3

As the scope of the variables is the **body**, The following code causes an error, because the variable **i** is not defined in the scope of the variable **j**.

(let ((i 1) (j (+ i 2)))

(\* i j))

;Error

The **let\*** expression is available to refer variables which is defined in the same binding form. Actually, the **let\*** expression is a syntax sugar of nested letexpressions.

(let\* ((i 1) (j (+ i 2)))

(\* i j))

;Value: 3

Example 3: A function quadric-equation that calculates the answers of quadratic equations.   
It takes three coefficient a, b, c (a x2 + b x + c = 0) as arguments and returns the list of answers in real numbers. The answers can be calculated without useless calculations by using let expressions step by step.

;;;The scopes of variables **d**,**e**, and **f** are the regions with the same background colors.

(define (quadric-equation a b c)

(if (zero? a)

'error ; 1

(let ((d (- (\* b b) (\* 4 a c)))) ; 2

(if (negative? d)

'() ; 3

(let ((e (/ b a -2))) ; 4

(if (zero? d)

(list e)

(let ((f (/ (sqrt d) a 2))) ; 5

(list (+ e f) (- e f)))))))))

(quadric-equation 3 5 2) ; solution of 3x2+5x+2=0

;Value 12: (-2/3 -1)

The function behaves like as follows:

1. If the 2nd order coefficient (**a**) is 0, it returns 'error.
2. If a ≠ 0, it binds the value of the discriminant (b2 - 4ac) to a variable **d**.
3. If **d** is negative, it returns '().
4. If not, it binds -b/2a to a variable **e**.
5. If **d** is zero, it returns a list consisting **e**.
6. If **d** is positive, it binds √(d/2a) to a variable **f** and returns a list of (+ e f) and (- e f).

Actually let expression is a syntax sugar of lambda expression:

(let ((p1 v1) (p2 v2) ...) exp1 exp2 ...)

⇒

((lambda (p1 p2 ...)

exp1 exp2 ...) v1 v2)

As the lambda expressions is function definition, it makes a scope of variables.

**Exercise 1**

Write a function described at [exercise 1 in chapter 4](http://www.shido.info/lisp/scheme4_e.html#practice) with one function,  
which means that writing a function to calculate flying distances with initial velocity of **v** and angle to the surface of **a**.

**2. Summary**

I have explained the **let** expression in this chapter. The let is a syntax sugar of lambda expression. The scope of variables is defined by using let or lambdaexpressions. In Scheme, the scope is made as written in the source code, which is called the **lexical closure**.

I will explain about repetition in the next chapter.

**Answers for Exercises**

**Answer 1**

(define (throw v a)

(let ((r (/ (\* 4 a (atan 1.0)) 180)))

(/ (\* 2 v v (cos r) (sin r)) 9.8)))