Lab 12: Macros

WWSD (required)

Q1: WWSD: Macros

在命令行中输入python3 scheme,然后在scheme中输入以下指令，思考为什么会有这样的运行结果

```
scm> +
#[+]
scm> list
#[list]
```

```
scm> +
______

scm> list
______

scm> (define-macro (f x) (car x))
______

scm> (f (2 3 4)) ; type SchemeError for error, or Nothing for nothing
______

scm> (f (+ 2 3))
______

scm> (define x 2000)
______

scm> (f (x y z))
______

scm> (f (list 2 3 4))
______

scm> (f (quote (2 3 4)))
______

scm> (define quote 7000)
______

scm> (f (quote (2 3 4)))
______
```

```
scm> (define-macro (g x) (+ x 2))
______

scm> (g 2)
______

scm> (g (+ 2 3))
______

scm> (define-macro (h x) (list '+ x 2))
______

scm> (h (+ 2 3))
______
```

```
scm> (define-macro (if-else-5 condition consequent) `(if ,condition ,consequent 5))
______

scm> (if-else-5 #t 2)
______

scm> (if-else-5 #f 3)
______

scm> (if-else-5 #t (/ 1 0))
______

scm> (if-else-5 #f (/ 1 0))
______

scm> (if-else-5 (= 1 1) 2)
______
```

Q2: WWSD: Quasiquote

```
scm> '(1 x 3)
______

scm> (define x 2)
______

scm> `(1 x 3)
______

scm> `(1 ,x 3)
______

scm> '(1 ,x 3)
______

scm> `(,1 x 3)
______

scm> `,(+ 1 x 3)
______

scm> `(1 (,x) 3)
______

scm> `(1 ,(+ x 2) 3)
______

scm> (define y 3)
______

scm> `(x ,(* y x) y)
______

scm> `(1 ,(cons x (list y 4)) 5)
______
```

Required Problems

Q3: Repeatedly Cube

Implement the following function, which cubes the given value x some number n times, based on the given skeleton.

```
(define (repeatedly-cube n x)
  (if (zero? n)
    x
    (let
      (_________________________)
      (* y y y))))
```

Q4: Scheme def

Implement def , which simulates a python def statement, allowing you to write code like (def f(x y) (+ x y)) .

The above expression should create a function with parameters x and y , and body (+ x y) , then bind it to the name f in the current frame.

> Note: the previous is equivalent to (def f (x y) (+ x y)) .

```
(define-macro (def func bindings body)
  'YOUR-CODE-HERE)
```

Q5: Switch

Define the macro switch , which takes in an expression expr and a list of pairs, cases , where the first element of the pair is some *value* and the second element is a single expression. switch will evaluate the expression contained in the list of cases that corresponds to the value that expr evaluates to.

```
scm> (switch (+ 1 1) ((1 (print 'a))
                      (2 (print 'b))
                      (3 (print 'c))))
b
```

You may assume that the value expr evaluates to is always the first element of one of the pairs in cases . Additionally, it is ok if your solution evaluates expr multiple times.

```
(define-macro (switch expr cases)
  'YOUR-CODE-HERE
)
```

将Q3,Q4，Q5的代码整理成一个scheme文件，命名方式:学号.scm(例如：10086.scm)，发送到sicp@foxmail.com，截至日期12月18号晚上9点