Example of REAL Macro

```
(match some-value (literally-hitler)
 ((literally-hitler . rest) ; First element is literally Hitler.
  (error "Found the Nazi"))
 (((a b) second . rest) ; First element is a two-element list.
  (display a))
 ((first second . rest) ; It's a list with at least two elements.
    (display (list first second)))
 (else #f))
```

Code For Macro

```
(require 'macro)

(define (maybe-car obj fail-value)
  (if (pair? obj)
      (car obj)
      fail-value))

(define (maybe-cdr obj fail-value)
  (if (pair? obj)
      (cdr obj)
      fail-value))

(define exists-in?
  (lambda (ele lis)
    (cond ((null? lis) #f)
          ((equal? ele (car lis)) #t)
          (else (exists-in? ele (cdr lis))))))

(define-syntax match-pattern
  (syntax-rules ()
    ;; No pattern. Matches if the match-value is null.
    ((_ () literals match-value fail-value success-expr)
     (if (null? match-value)
            success-expr
            fail-value))
    ;; Notice there are TWO pattern-matches going on: One at compile-time via
    ;; syntax-rules, and another at runtime, being done with cond forms
    ;; and comparison with the 'fail-value to detect failures deeper in the
    ;; pattern.
    ;;
    ;; This case matches when the first element of the pattern is a list.
    ;; It generates code that matches the match-value only if its first element
    ;; is also a list.
    ((_ ((hhd . htl) . tl) literals match-value fail-value success-expr)
     (cond ((eq? match-value fail-value)
              fail-value)
       ;; Macros are allowed to expand into instances of themselves.
             (else (match-pattern (hhd . htl) literals (maybe-car match-value fail-value)
                                               fail-value
                         (match-pattern tl literals (maybe-cdr match-value fail-value) fail-value
                                                  success-expr)))))
    ;; Matches if the pattern itself is a list. hd, short for "head", is a
    ;; variable that will be bound to the first element of the match-value if it's
    ;; a list. If it's not a list, (maybe-car) will cause hd to be bound to the fail-value.
    ;;
    ;; Also, the match-value may already be the fail-value due to occurrences at a shallower
    ;; level in the pattern. If this happens, then this code won't bother to delve any deeper.
    ((_ (hd . tl) literals match-value fail-value success-expr)
     (cond ((eq? match-value fail-value)
              fail-value)
             ((exists-in? 'hd 'literals)
              (if (eq? (maybe-car match-value fail-value) 'hd)
                      (match-pattern tl literals (maybe-cdr match-value fail-value)
                                      fail-value success-expr)
                      fail-value))
             (else
              (let ((hd (maybe-car match-value fail-value)))
                (if (eq? hd fail-value)
                         fail-value
                         (match-pattern tl literals (maybe-cdr match-value fail-value)
                                               fail-value success-expr))))))
    ;; The pattern doesn't have to be a list. If it's not, it'll be bound to the
    ;; whole match-value. Control can also reach here if the non-list pattern
    ;; is in the cdr position of a larger pattern.
    ((_ non-list literals match-value fail-value success-expr)
     (cond ((eq? match-value fail-value)
              fail-value)
             ((exists-in? 'non-list 'literals)
              (if (eq? 'non-list match-value)
                      success-expr
                      fail-value))
             (else (let ((non-list match-value)) success-expr))))))
```