Here pattern `(x y)` matches any two-element list, regardless of the types of these elements. Pattern variables *x* and *y* are bound to, respectively, the first and second element of *l*.

Patterns can be composed, and nested. For instance, `...` (ellipsis) means that the previous pattern may be matched zero or more times in a list:

```
(match lst
  (((heads tails ...) ...)
   heads))
```

This expression returns the first element of each list within *lst*. For proper lists of proper lists, it is equivalent to `(map car lst)`. However, it performs additional checks to make sure that *lst* and the lists therein are proper lists, as prescribed by the pattern, raising an error if they are not.

Compared to hand-written code, pattern matching noticeably improves clarity and conciseness—no need to resort to series of `car` and `cdr` calls when matching lists, for instance. It also improves robustness, by making sure the input *completely* matches the pattern—conversely, hand-written code often trades robustness for conciseness. And of course, `match` is a macro, and the code it expands to is just as efficient as equivalent hand-written code.

The pattern matcher is defined as follows:

`match` *exp clause1 clause2 . . .* [Scheme Syntax]

Match object *exp* against the patterns in *clause1 clause2 . . .* in the order in which they appear. Return the value produced by the first matching clause. If no clause matches, throw an exception with key `match-error`.

Each clause has the form `(pattern body1 body2 ...)`. Each *pattern* must follow the syntax described below. Each body is an arbitrary Scheme expression, possibly referring to pattern variables of *pattern*.

The syntax and interpretation of patterns is as follows:

```
        patterns:                       matches:

pat ::= identifier                      anything, and binds identifier
      | _                               anything
      | ()                              the empty list
      | #t                              #t
      | #f                              #f
      | string                          a string
      | number                          a number
      | character                       a character
      | 'sexp                           an s-expression
      | 'symbol                         a symbol (special case of s-expr)
      | (pat_1 ... pat_n)               list of n elements
      | (pat_1 ... pat_n . pat_{n+1})   list of n or more
      | (pat_1 ... pat_n pat_n+1 ooo)   list of n or more, each element
                                          of remainder must match pat_n+1
      | #(pat_1 ... pat_n)              vector of n elements
      | #(pat_1 ... pat_n pat_n+1 ooo)  vector of n or more, each element
                                          of remainder must match pat_n+1
      | #&pat                           box
      | ($ record-name pat_1 ... pat_n) a record
      | (= field pat)                   a ``field'' of an object
      | (and pat_1 ... pat_n)           if all of pat_1 thru pat_n match
      | (or pat_1 ... pat_n)            if any of pat_1 thru pat_n match
      | (not pat_1 ... pat_n)           if all pat_1 thru pat_n don't match
      | (? predicate pat_1 ... pat_n)   if predicate true and all of
                                          pat_1 thru pat_n match
      | (set! identifier)               anything, and binds setter
      | (get! identifier)               anything, and binds getter
      | `qp                             a quasi-pattern
      | (identifier *** pat)            matches pat in a tree and binds
                                        identifier to the path leading
                                        to the object that matches pat

ooo ::= ...                             zero or more
      | ___                             zero or more
      | ..1                             1 or more

        quasi-patterns:                 matches:

qp  ::= ()                              the empty list
      | #t                              #t
      | #f                              #f
      | string                          a string
      | number                          a number
      | character                       a character
      | identifier                      a symbol
      | (qp_1 ... qp_n)                 list of n elements
      | (qp_1 ... qp_n . qp_{n+1})      list of n or more
      | (qp_1 ... qp_n qp_n+1 ooo)      list of n or more, each element
                                          of remainder must match qp_n+1
      | #(qp_1 ... qp_n)                vector of n elements
      | #(qp_1 ... qp_n qp_n+1 ooo)     vector of n or more, each element
                                          of remainder must match qp_n+1
      | #&qp                            box
      | ,pat                            a pattern
      | ,@pat                           a pattern

             patterns:                       matches:

     pat ::= identifier                      anything, and binds identifier
           | _                               anything
           | ()                              the empty list
           | #t                              #t
           | #f                              #f
           | string                          a string
           | number                          a number
           | character                       a character
           | 'sexp                           an s-expression
           | 'symbol                         a symbol (special case of s-expr)
           | (pat_1 ... pat_n)               list of n elements
           | (pat_1 ... pat_n . pat_n+1)   list of n or more
           | (pat_1 ... pat_n pat_n+1 ooo)   list of n or more, each element
                                               of remainder must match pat_n+1
           | #(pat_1 ... pat_n)              vector of n elements
           | #(pat_1 ... pat_n pat_n+1 ooo)  vector of n or more, each element
                                               of remainder must match pat_n+1
           | #&pat                           box
           | ($ record-name pat_1 ... pat_n) a record
           | (= field pat)                   a ``field'' of an object
           | (and pat_1 ... pat_n)           if all of pat_1 thru pat_n match
           | (or pat_1 ... pat_n)            if any of pat_1 thru pat_n match
           | (not pat_1 ... pat_n)           if all pat_1 thru pat_n don't match
           | (? predicate pat_1 ... pat_n)   if predicate true and all of
                                               pat_1 thru pat_n match
           | (set! identifier)               anything, and binds setter
           | (get! identifier)               anything, and binds getter
           | `qp                             a quasi-pattern
           | (identifier *** pat)            matches pat in a tree and binds
                                             identifier to the path leading
                                             to the object that matches pat
```

patterns:

`identifier`
anything, and binds identifier

`_` anything

`()` the empty list

`#t` #t

`#f` #f

`string` a string

`number` a number

`character`
a character

`'sexp` an s-expression

`'symbol` a symbol (special case of s-expr)

`(`*`pat_1 ... pat_n`*`)`
list of n elements

`(pat_1... pat_n . pat_n+1)`
list of n or more

`(pat_1... pat_n pat_n+1 ooo)`
list of n or more, each element of remainder must match pat_n+1

`#(pat_1... pat_n)`
vector of n elements

`#(pat_1... pat_n pat_n+1 ooo)`
vector of n or more, each element of remainder must match pat_n+1

`#&pat` box

`($ record-name pat_1 ... pat_n)`
a record

`(= field pat)`
a “field” of an object

`(and pat_1 ... pat_n)`
if all of pat_1 thru pat_n match

`(or pat_1 ... pat_n)`
if any of pat_1 thru pat_n match

`(not pat_1 ... pat_n)`
if all pat_1 thru pat_n don’t match

`(? predicate pat_1 ... pat_n)`
if predicate true and all of pat_1 thru pat_n match

`(set! identifier)`
anything, and binds setter

`(get! identifier)`
anything, and binds getter

`` `qp `` a quasi-pattern

`(identifier *** pat)`
matches pat in a tree and binds identifier to the path leading to the object that matches pat

ooo:

`...` zero or more

`___` zero or more

`..1` 1 or more

quasi-patterns:

`()` the empty list

`#t` `#t`