A Scheme Expression is a Scheme List

Scheme programs consist of expressions, which can be:

- Primitive expressions: 2 3.3 true + quotient
- Combinations: (quotient 10 2) (not true)

The built-in Scheme list data structure (which is a linked list) can represent combinations

```
> (list 'quotient 10 2)
(scm> (eval (list 'quotient 10 2)))
5
```

In such a language, it is straightforward to write a program that writes a program

```
> (twice (print 2))
2
2
```

Evaluation procedure of a macro call expression:
- Evaluate the operator sub-expression, which evaluates to a macro
- Call the macro procedure on the operand expressions without evaluating them first
- Evaluate the expression returned from the macro procedure

```
> (for x '2 3 4 5) (* x x)
4 9 16 25
```

Macros Perform Code Transformations

A macro is an operation performed on the source code of a program before evaluation.

Macros exist in many languages, but are easiest to define correctly in a language like Lisp.

Scheme has a `define-macro` special form that defines a source code transformation

```
(define-macro (twice expr)
  (list 'begin expr expr))
```

```
> (twice (print 2))
2
2
```

Discussion Question

Define a macro that evaluates an expression for each value in a sequence

```
(define-map fn vals)
  (if (null? vals)
    ()
    (cons (fn (car vals))
      (map fn (cdr vals))))
```

```
> (map (lambda (x) (* x x)) '2 3 4 5)
4 9 16 25
```

For Macro

```
(define-macro (for sym vals expr)
  (list 'map
        (list 'lambda
          (list sym vals) expr)
        vals)
  )
```

```
> (for x '2 3 5) (+ x 1)
4 6 8 10
```

(Sdemo)
Quasi-Quotation

(Democ)