Macros

Expressions

Announcements

Discussion Question: Pythagorean Theorem

Quick quasiquotation review: `(+(* 2 3) 1)` evaluates to (+ 6 1)

Add `define` and `, in some blanks so that the second expression evaluates to (+ (* a a) (+ b b))

```
(define (square-expr term) `(* ,term ,term))
```

```
(+ (+ (square-expr `a) (square-expr `b)))
```

(Demo)
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))
```

```
(scm> (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

(Demo)

Discussion Question

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

```
(scm> (map (lambda (x) (+ x x)) (2 3 4 5)))
(4 9 16 25)
```

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

```
(scm> (for x (2 3 4 5) (+ x x))
(4 9 16 25)
```

(Demo)
Trace

Trace Recursive Calls

```python
def trace(fn):
    def traced(n):
        print(f'{fn.__name__}({n})')
        return fn(n)
    return traced

@trace
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n - 1)
```

```plaintext
>>> fact(5)
120
```

```scm
(scm> (fact 5))
120
```

Demo