

## Discussion Question: Pythagorean Theorem

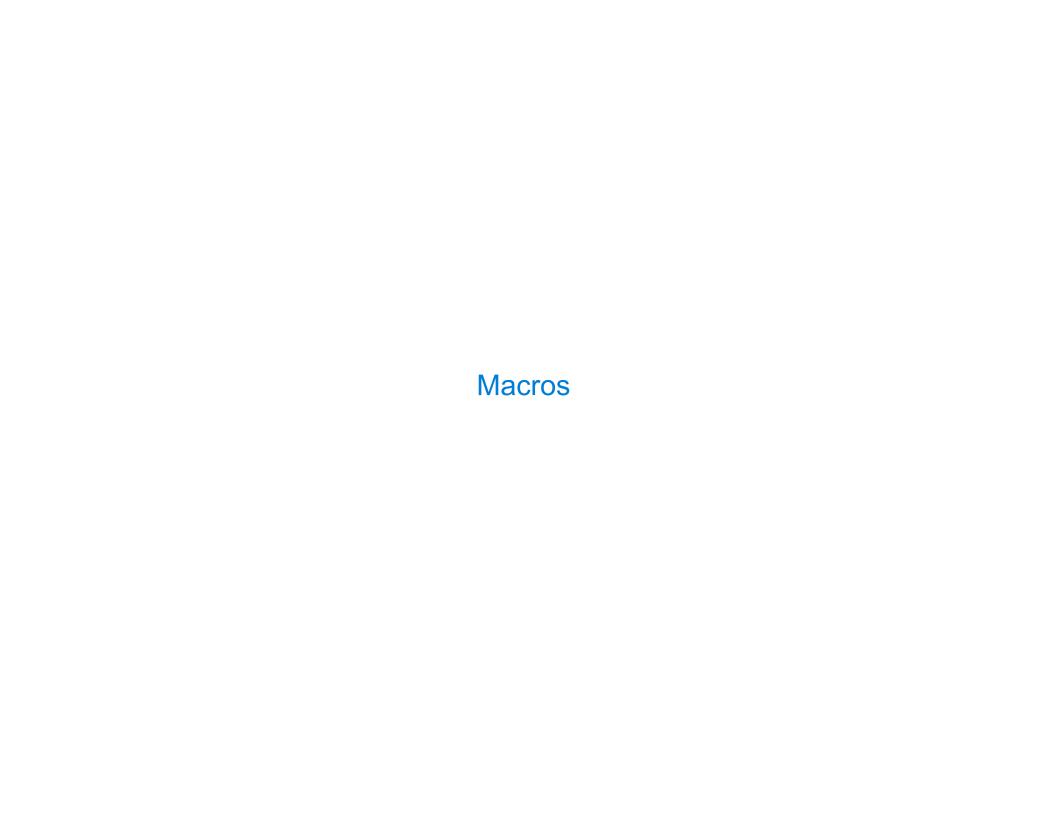
```
Quick quasiquotation review: `(+ ,(* 2 3) 1) evaluates to (+ 6 1)

Add ` 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

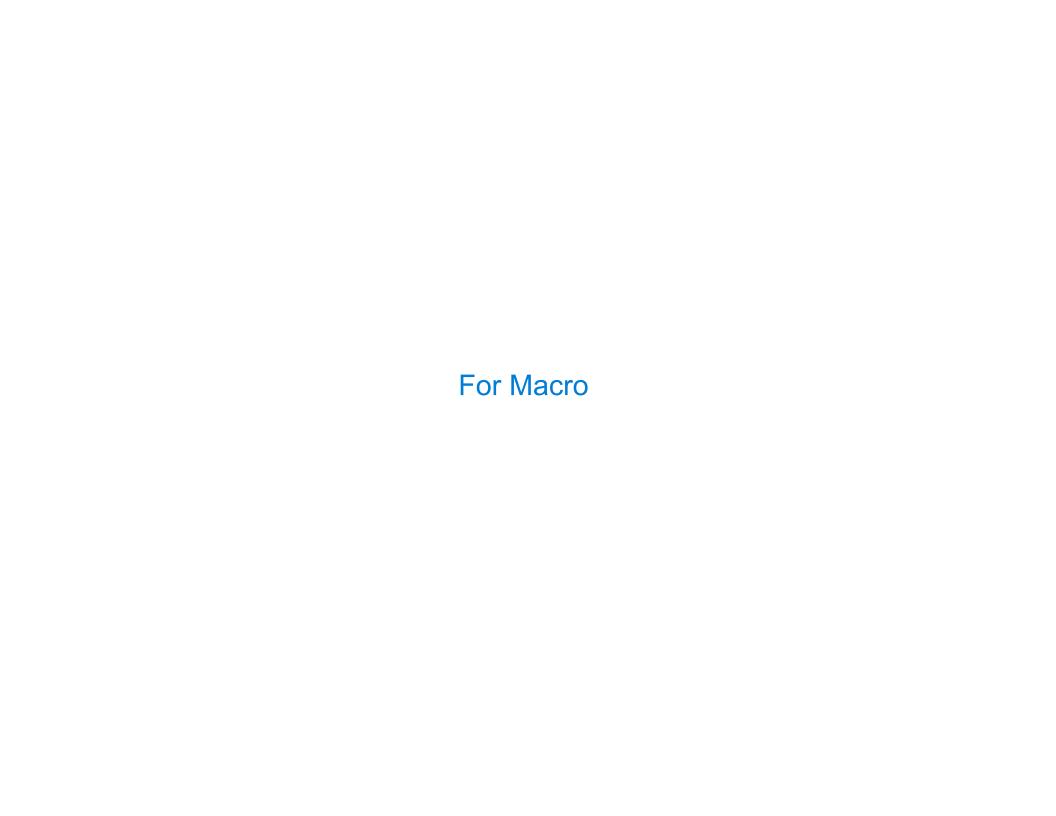
Macros exist in many languages, but are easiest to define correctly in a language like Lisp

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

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**

scm> (for x (2 3 4 5) (\* x x))

(4 9 16 25)

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

(Demo)



## **Tracing Recursive Calls**

```
def trace(fn):
                                               (define fact (lambda (n)
    def traced(n):
                                                 (if (zero? n) 1 (* n (fact (- n 1))))))
        print(f'{fn. name }({n})')
        return fn(n)
                                               (define original fact)
    return traced
                                               (define fact (lambda (n)
                                                          (print (list 'fact n))
@trace
                                                          (original n)))
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n - 1)
>>> fact(5)
                                               scm> (fact 5)
fact(5)
                                               (fact 5)
fact(4)
                                               (fact 4)
fact(3)
                                               (fact 3)
fact(2)
                                               (fact 2)
fact(1)
                                               (fact 1)
                                               (fact 0)
fact(0)
                                                                                         (Demo)
120
                                               120
```