The “Beauty” of Scheme: Programs as Proofs

- *Q*: Is **`'(abc 123)`** a list?
- *A*: Yes
- *Proof*:
  - **`'(abc 123) = (cons 'abc (cons 123 '()))`**
  - **`'()`** is a list, so **`(cons 123 '())`** is also a list
  - **`(cons 123 '())`** is a list, so

    `(cons 'abc (cons 123 '()))`

    is also a list, *Q.E.D.*

The Beauty of Scheme: Programs as Definitions

Factorial: definition

Base-case: *fac(0) = 1*
Induction: *fac(n) = n * fac(n-1)*

Factorial: program

```
(define fac
  (lambda(n)
    (if (eq? n 0)
        1
        (* n (fac (- n 1))))))
```

The Power of Scheme: Functions as *First-Class Objects*

```
; apply a function to each number in a list
(define map
  (lambda (fn ln)          ; function, list of numbers
    (if (null? ln)         ; null? means equals '()
        '()
        (cons
           (fn (car ln))    ; apply fn to first elmt
           (map fn (cdr ln))))))  ; recur on rest



> (map square '(1 2 3 4 5))
(1 4 9 16 25)
```

Shortcuts: **cond**

```
(define (sign x)
  (if (< x 0)
      -1
      (if (> x 0)
          1
          0)))

(define (sign x)
  (cond ((< x 0) -1)
        ((> x 0)  1)
        (else 0))) ; sensible default
```

Shortcuts: `list-ref`

If **`ls`** is a list then

```
(car ls)              = (list-ref ls 0)
(car (cdr ls))        = (list-ref ls 1)
(car (cdr (cdr ls)))  = (list-ref ls 2)
```

(Like Java/C++ arrays)

Shortcuts: `car, cdr`

```
(caar ls) = (car (car ls))
(cadr ls) = (car (cdr ls))
(cdadr ls) = (cdr (car (cdr ls)))
```

(etc.)

Comparison with Python/Java/C++

- *Binding* instead of assignment:

  `(define a 5)` *vs.* `a = 5;`

Comparison with Python/Java/C++

- Recursion instead of loops:

```
int sum(int n) { // sum from 0 to n
  int s = 0;
  for (int i=1; i<=n; ++i) {
    s += i;
  }
  return s;
}
```

```
(define sum
    (lambda (n)
      (if (eq? n 0)
        0
        (+ n (sum (- n 1))))))
```

Interpreter: The Read/Eval/Print Loop

```
> 1
1
> (+ 3 2)
5
> ((lambda(x) (* x x)) 4)
16
```