Our common framework

Goal: eliminate superficial differences
- Makes comparisons easy
- Differences that remain must be important!

No new language ideas.

Imperative programming with an IMPerative CORE:
- Has features found in most languages (loops and assignment)
- Trivial syntax (from LISP)
Idea of LISP syntax

Parenthesized prefix syntax:

- Names and numerals are basic atoms
- Other constructs bracketed with ( . . . ) or [ . . . ]
  (Possible keyword after opening bracket)

Examples:

(+ 2 2)
(if (isbound? x rho) (lookup rho x) (error 99))

(For now, we use just the round brackets)
Impcore structure

Two syntactic categories: expressions, definitions

No statements!—expression-oriented (compositional)

(if e₁ e₂ e₃)
(while e₁ e₂)
(set x e)
(begin e₁ ... eₙ)
(f e₁ ... eₙ)

Evaluating e has value, may have side effects

Functions f named (e.g., + - * / = < > print)

The only type of data is “machine integer”
(deliberate oversimplification)
Syntactic structure of Impcore

An Impcore program is a sequence of definitions

\[(\text{define mod (m n)} \ (- \ m \ (* \ n \ (/ \ m \ n))))\]

Compare

```c
int mod (int m, int n) {
    return m - n * (m / n);
}
```
Impcore variable definition

Example

(val n 99)

Compare

int n = 99;
Concrete syntax for Impcore

Definitions and expressions:

\[
def ::= (define f (x1 \ldots x_n) \text{exp}) ;; "true" \text{defs} \\
| (val x \text{exp}) \\
| \text{exp} \\
| (use \text{filename}) ;; "extended" \text{defs} \\
| (check-expect \text{exp1} \text{exp2}) \\
| (check-assert \text{exp}) \\
| (check-error \text{exp})
\]

\[
\text{exp} ::= \text{integer-literal} \\
| \text{variable-name} \\
| (set x \text{exp}) \\
| (if \text{exp1} \text{exp2} \text{exp3}) \\
| (while \text{exp1} \text{exp2}) \\
| (begin \text{exp1} \ldots \text{expn}) \\
| (function-name \text{exp1} \ldots \text{expn})
\]
(define even? (n) (= (mod n 2) 0))

(define 3n+1-sequence (n) ; from Collatz
  (begin
    (while (!= n 1)
      (begin
        (println n)
        (if (even? n)
          (set n (/ n 2))
          (set n (+ (* 3 n) 1)))))
    n))