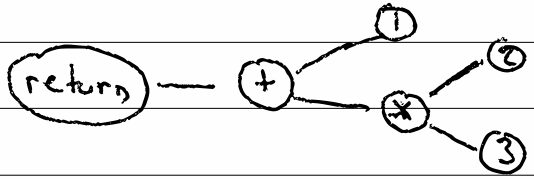
1-1/ why?

- implementing PLs
- implement lang features
- understanding what progs mean

What is a program?

" return 1 + 2 * 3; " — a 17 character string

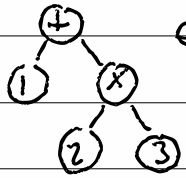


— a 6 nodetree

structured code editors — Scratch

1-2/ J_0 $e := v \mid (+\ e\ e) \mid (*\ e\ e)$

$v :=$ numbers



$\in J_0.e$

$= (+\ 1\ (*\ 2\ 3))$

Lisp
Scheme
Racket
Clojure

```
interface JExpr () {}
class JNum implements JExpr { int n;  JNum(int n) { this.n = n; } }
class JPlus implements JExpr { JExpr lhs, rhs;  JPlus(JExpr lhs, rhs)
class JMult implement JExpr { JExpr lhs, rhs;  .... }
= new JPlus( new JNum (1),
     new JMult ( new JNum (2), new JNum (3)));
```

1-3/① pp : e → string

② pp n = arg_str n

③ pp (+ lhs rhs) = "(+ " ++ pp lhs ++ " " ++ pp rhs ++ ")"

pp (* lhs rhs) = "(* " ++ pp lhs ++ " " ++ pp rhs ++ ")"

① interface JExpr { public String pp(); }

② class JNum { public String pp() { return n.itos(); } }

③ class JPlus { public String pp() {

return "(+ " + lhs.pp() + " " + rhs.pp() + ")"; } }

1-4) The most important thing is what a program **means**

big-step semantics / interpreter

interface IExpr { public int interp(); }

$interp : e \rightarrow v$

$interp\ n = n$

$interp\ (+\ e_L\ e_R) = interp\ e_L + interp\ e_R$

① $interp\ (\times\ e_L\ e_R) = interp\ e_L + interp\ e_R$

↓

```
class IMult {
    public int interp() {
        return this.lhs.interp() * this.rhs.interp(); } }
```

1-5/

JO	meaning
1	1
(+ 1 2)	3
(+ -1 1)	0
(x -1 8)	-8
(x 2 2)	4
(+ 1 (x 2 3))	7

examples / test suite

check ((new JPlus (new JNum(1), new JNum(2))), 3)

check (JExpr e, int eans)

if (e.interp() ≠ eans) then

error

1-6/ parser : string → e — complicated
Antlr, yacc/bison

reader : string → sexpr
desugar : sexpr → e
— less complicated

Sexpr se ::= string | num | (se ...)
atom
see site for 1-sexpr.c | cons se se
| null

(+ 1 (x 2 3)) ⇒ ["+", 1, ["x", 2, 3]]

1-7/ desugar : sexpr → e

desugar n = n

① desugar ["+", l, r] = (+ desugar(l) desugar(r))

desugar ["x", l, r] = (* desugar(l) desugar(r))

```
if ( se.length == 3 && se[0] == "+" ) {
  new JPlus ( desugar ( se[1] ), desugar ( se[2] ) );}
```

J₀ e ::= n | (+ e e) | (* e e)

J₀ surface := n | (+ e ...) | (* e ...)
| (- e) | (- e e)

1-8)

desugar ["-", e] = desugar ["*", -1, e]

desugar ["-", e_L, e_R] = desugar ["+", e_L, ["-", e_R]]

["+"] => [0]

(1) ["+", e_L, em ...] => (+ de(e_L)
de(["+", em ...])

["x"] => [1]

if (se.length >= 2 && se[0] == "+")
return new JPlus (desugar(se[1]),
desugar (new Cons (se[0],
se[2 ...])));

1-9/ J_1 $e ::= v \mid (e \ e \ \ldots) \mid (\text{if } e \ e \ e)$

$v ::= b$

$b ::=$ some set of constants

num | bool | prim

prim $::= + \mid * \mid - \mid / \mid \leq \mid < \mid = \mid > \mid \geq$

interp $v = v$

interp (if e_c e_t e_f) = interp e_k

where e_k = if interp e_c == false then e_f o.w. e_t

interp $(e_f \ e_{arg} \ldots)$ = $\delta(p, v_{arg} \ldots)$

where p = interp e_f ($\in$ prim)

v_{arg} = interp e_{arg}

$\delta(+, 1, 2) = 3$ $\quad \delta(<, 1, 3) = \text{true}$

$\delta(/, 1, 0) = \perp$ $\quad$ δ is operational fun that defines what prims mean