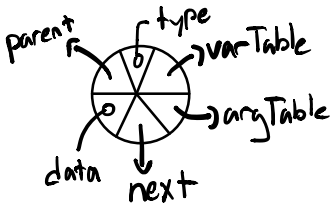
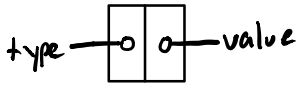
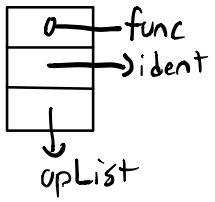
AST_NODE



AST_NUMBER



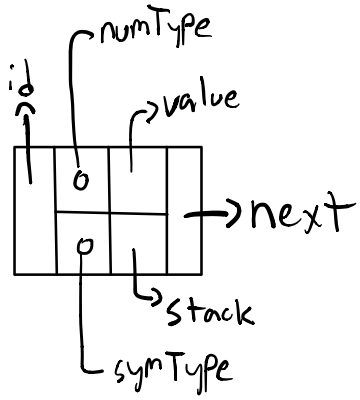
AST_FUNCTION



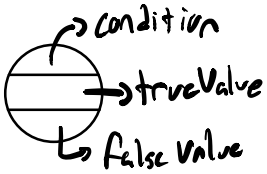
AST_SYMBOL

AST_SCOPE

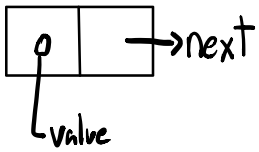
SYMBOL_TABLE_NODE

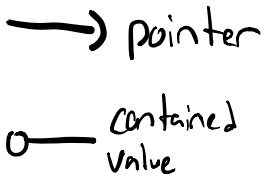


AST_CONDITIONAL



STACK_NODE





Define and evaluate custom function

- symbol table nodes can now define functions
- AST Nodes can now reference arguments for use by custom functions
- Symbol table nodes now specify a symbol type:
  - Variable (already implemented)
  - Lambda (function)
  - Argument (input for function)

- parse arg section
- parse function definition
- parse let section
- parse s_expr
- parse scope

0) look up function

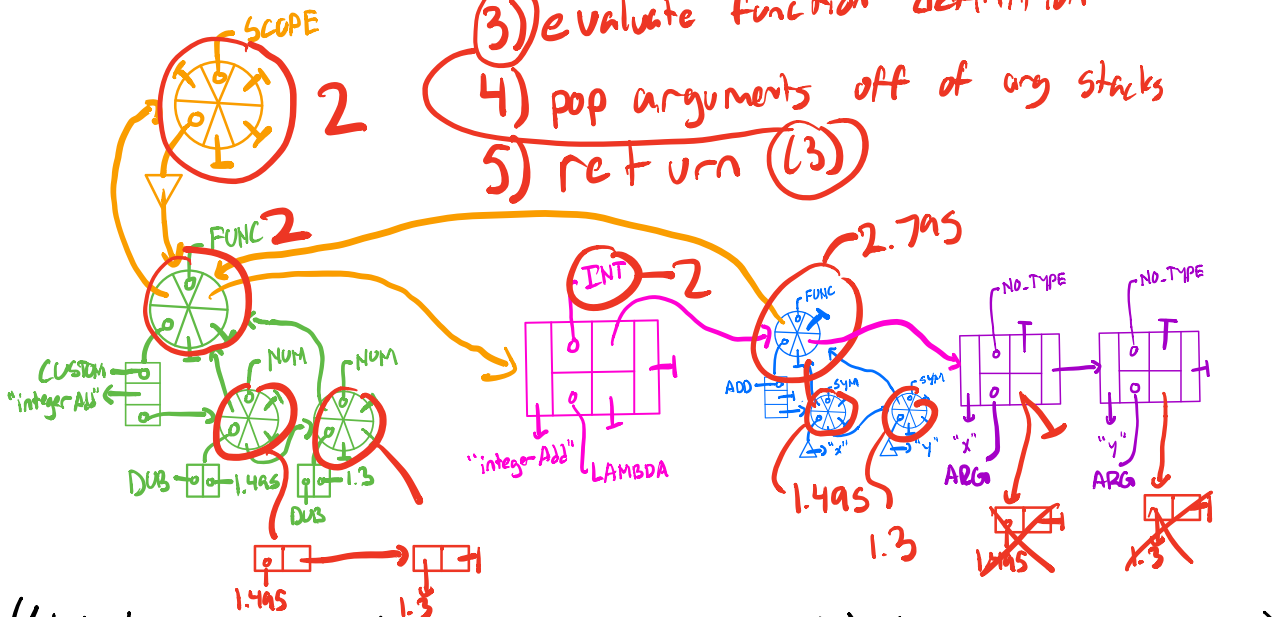
1) eval all operands

2) put operands on argument stacks

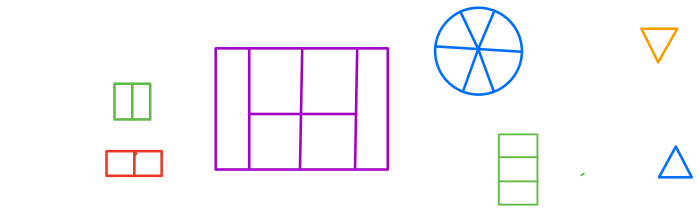
3) evaluate function definition

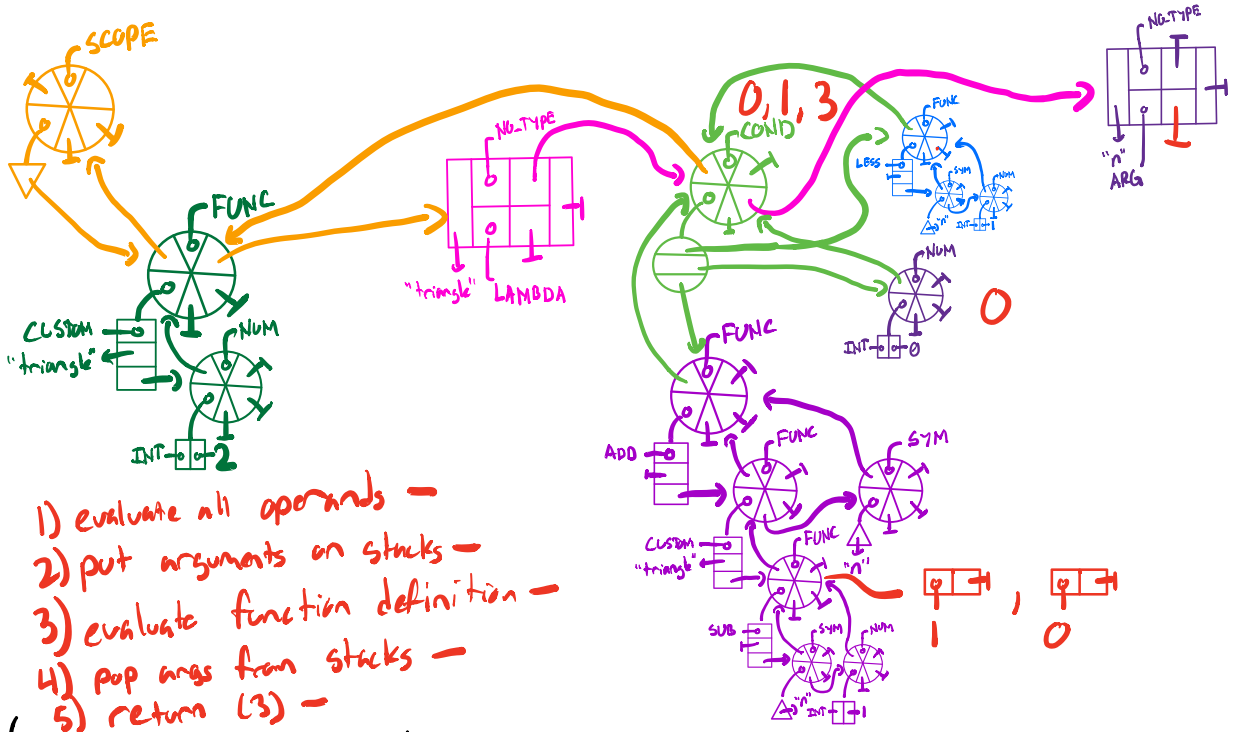
4) pop arguments off of arg stacks

5) return (3)



((let {int integer-Add lambda {x y} {add x y}}} {integer-Add 1.495 1.3 }))





1) evaluate all operands —
2) put arguments on stacks —
3) evaluate function definition —
4) pop args from stacks —
5) return (3) —

```
((let (triangle lambda (n) (cond ((less n 1) 0) ((add (triangle (sub n 1)) n)))) (triangle 2))
```

arg_elem ::= symbol

arg_list ::= arg_elem | arg_elem arg_list

arg_section ::= (arg_list)

f_expr ::= (FUNC s_expr_list)
| (SYMBOL s_expr_list)

(SYMBOL)

s_expr

s_expr ::= ~~let_section~~ s_expr

let_section ::= (let let_list)

({let {B lambda {a b c} {max a b c}} {r 1} {u 2} {h 3}} {B r u h})

0	1	2	3	4	5	6	7	8
0	1	1	2	3	5	8	13	21

if_stmt ::= if expr then stmt

| if expr then stmt else stmt