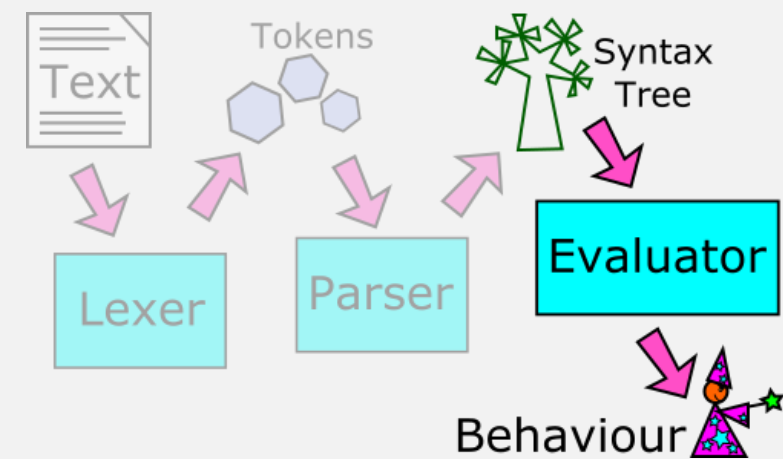


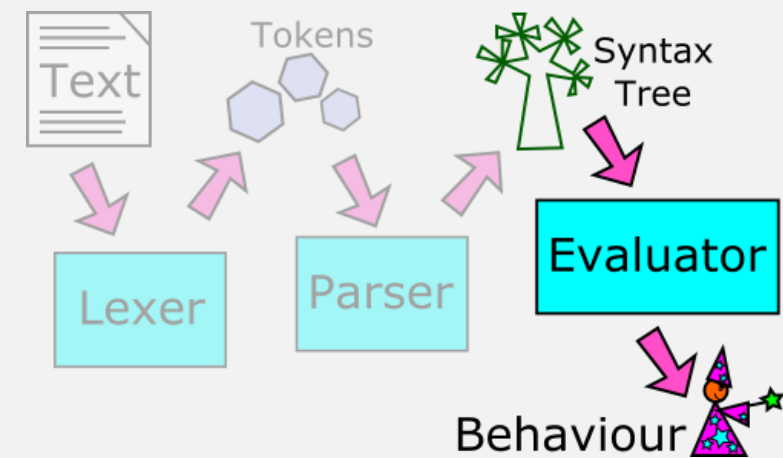
UMass Boston Computer Science  
**CS450 High Level Languages**  
**Interpreters and “eval”**

Thursday, April 2, 2026



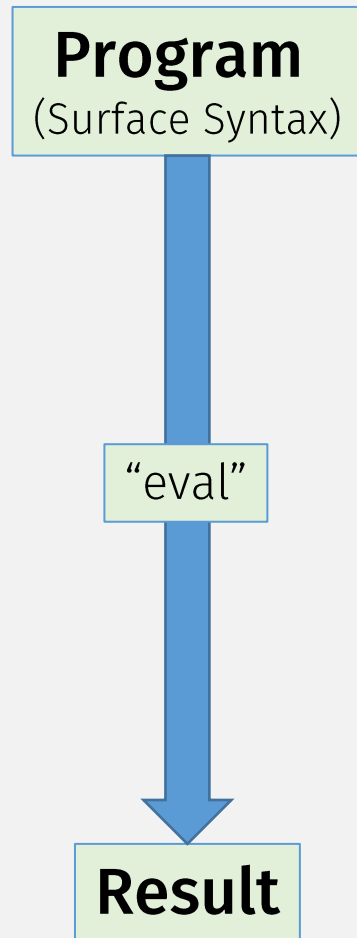
# Logistics

- HW 8 out
  - due: Tues 4/7, 11am EST

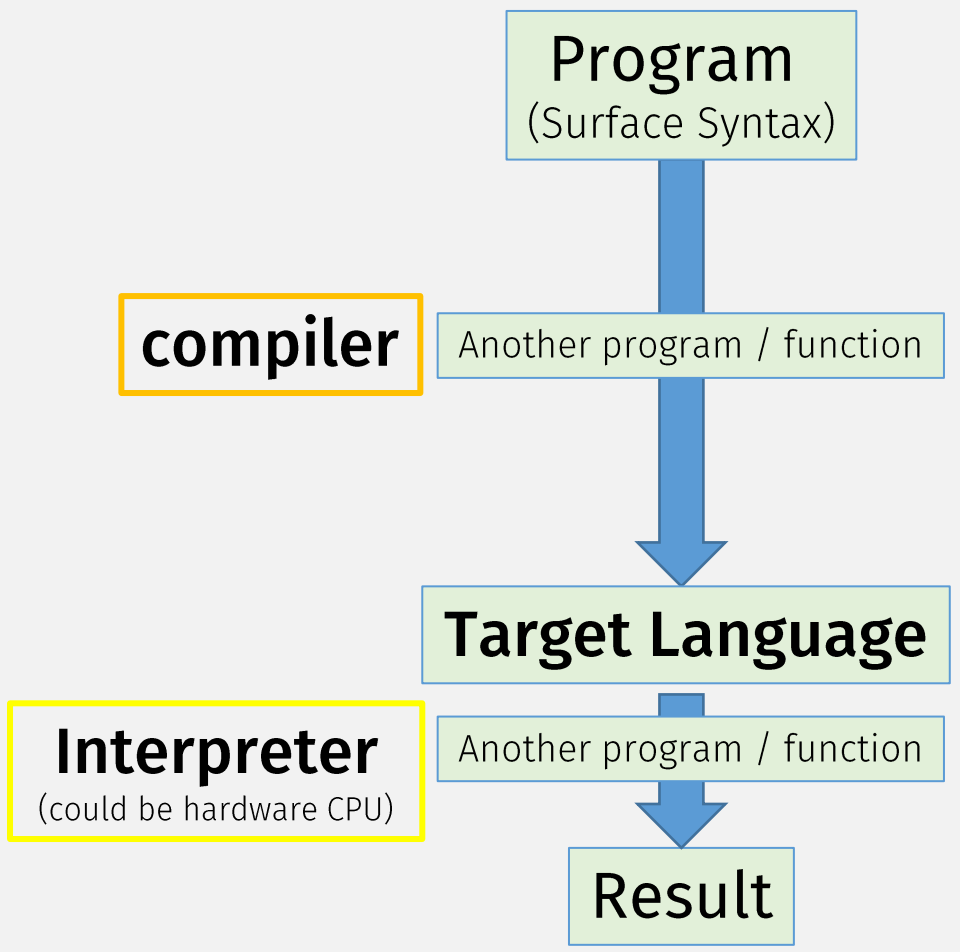


*Last Time*

# Giving Meaning to (i.e., Running) Programs



Last Time



More commonly, a high-level program is first compiled to a lower-level target language (and then interpreted)

*Last Time*

**compiler**

**Program**  
(Surface Syntax)



**Target Lang 1**



...



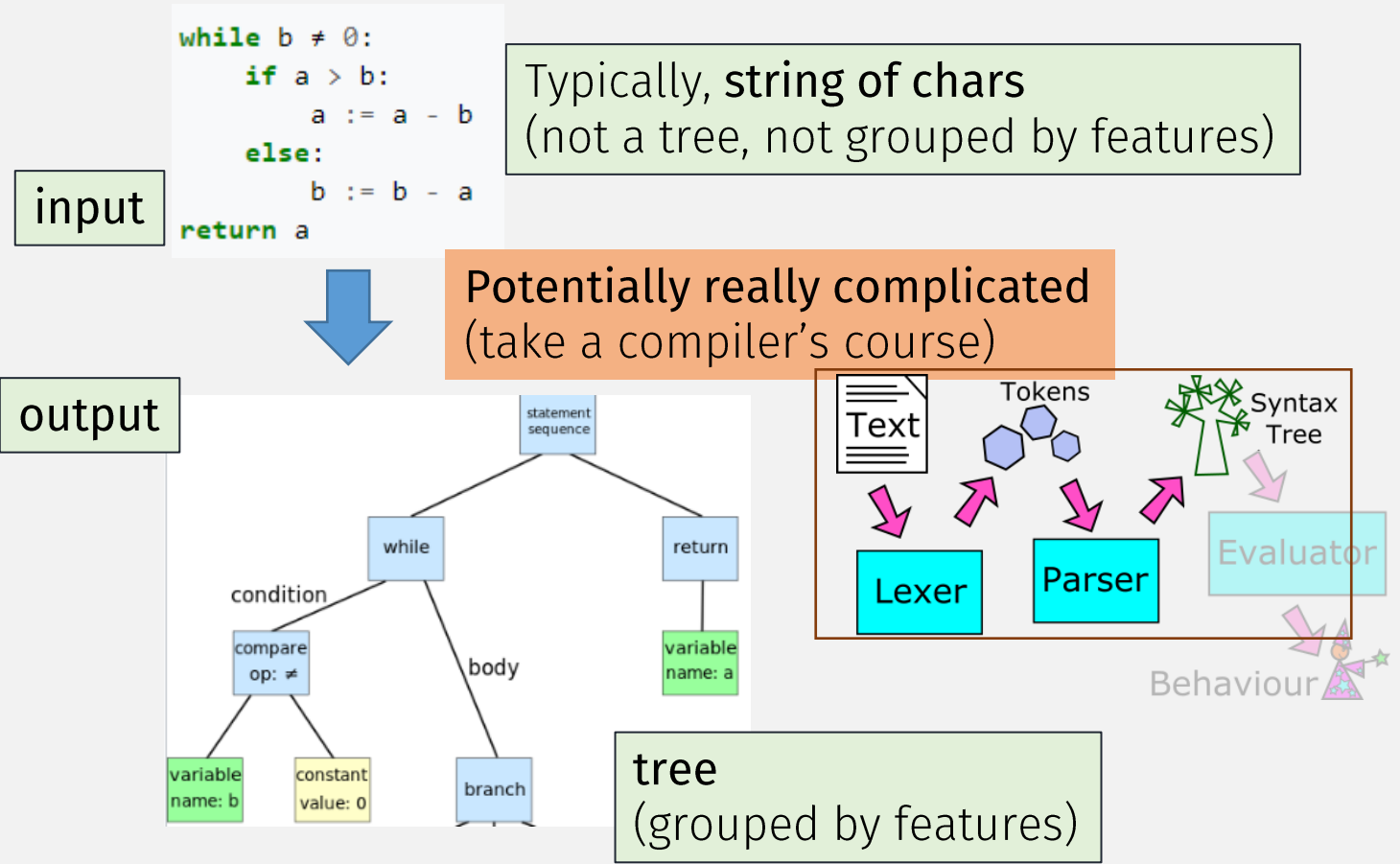
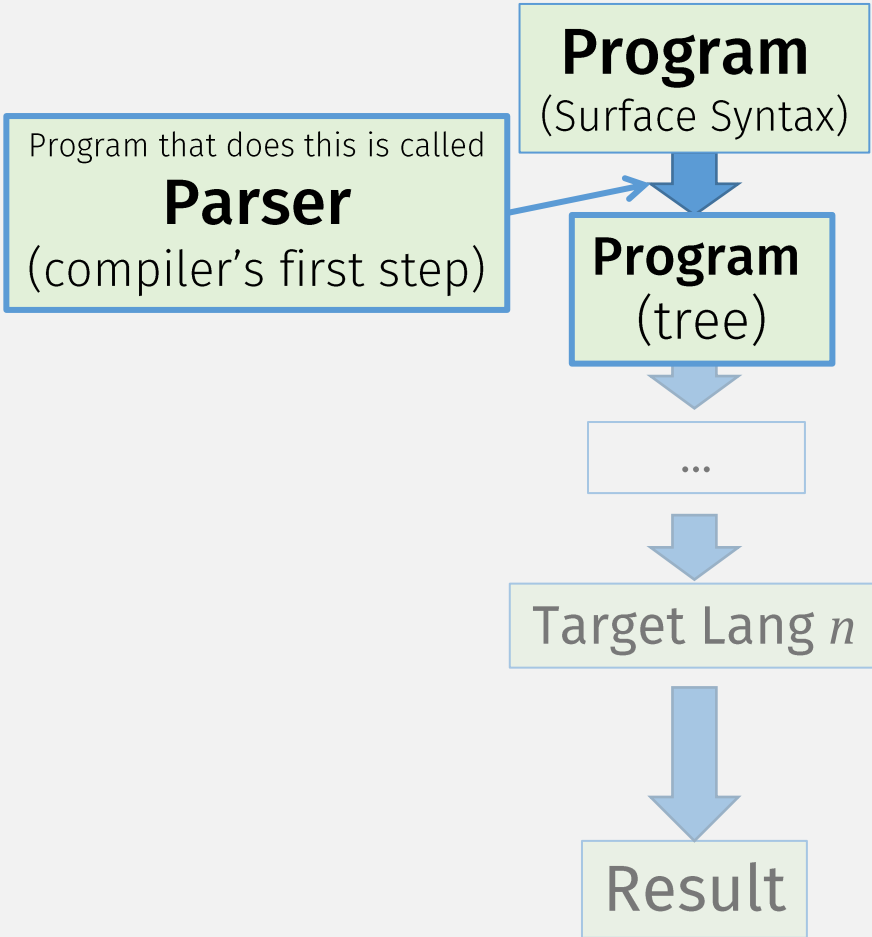
**Target Lang  $n$**



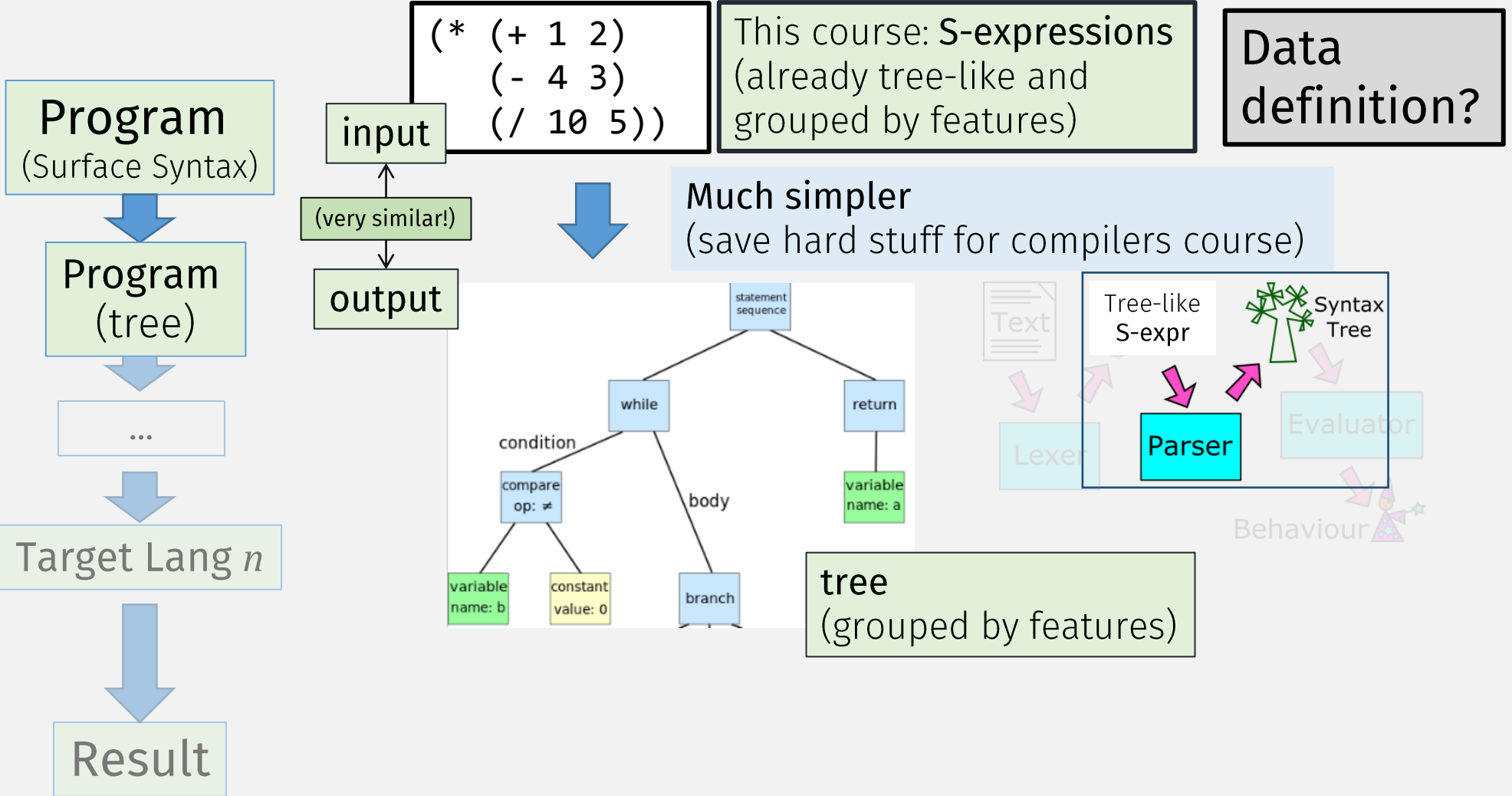
**Result**

**Compilers** often have multiple steps

# Parsing



# Parsing – This Course



```
;; A Program (Ssexpr) is a:  
;; - Number  
;; - `(Σ ,Program ,Program)  
;; - `(Π ,Program ,Program)
```

```
(* (+ 1 2)  
   (- 4 3)  
   (/ 10 5))
```

This course: S-expressions  
(already tree-like and  
grouped by features)

Data  
definition?

```
(define (prog-fn p)  
  (match p  
    [(? number?) ... ]  
    [ `(Σ ,x ,y)  
      ... (prog-fn x) ... (prog-fn y) ... ]  
    [ `(Π ,x ,y)  
      ... (prog-fn x) ... (prog-fn y) ... ]))
```

TEMPLATE?

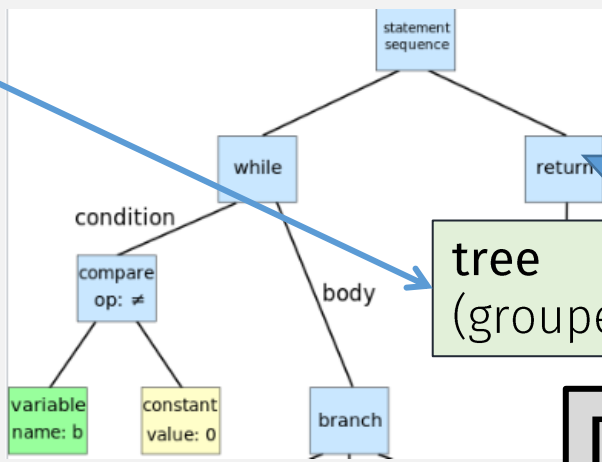
;; A Program (Program)  
;; - Number  
;; - `(Σ , Program Program)`  
;; - `(Π , ...)`

**Program**  
(Surface Syntax)

(\* (+ 1 2)  
(- 4 3)  
(/ 10 5))

This course: S-expressions  
(already tree-like and  
grouped by features)

**Abstract Syntax  
(Program) Tree (AST)**

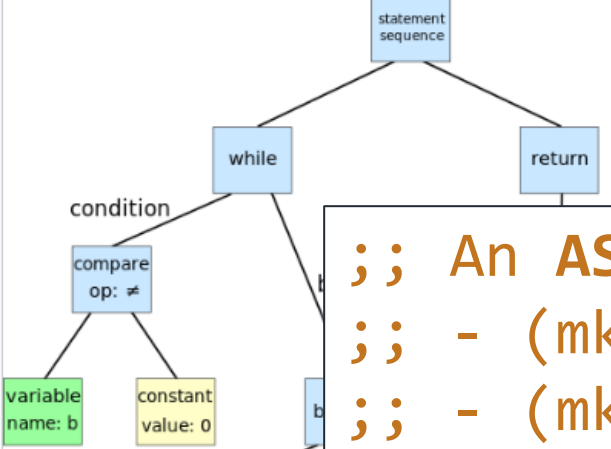


**tree**  
(grouped by features)

**Data  
definition?**



**Abstract Syntax  
(Program) Tree (AST)**

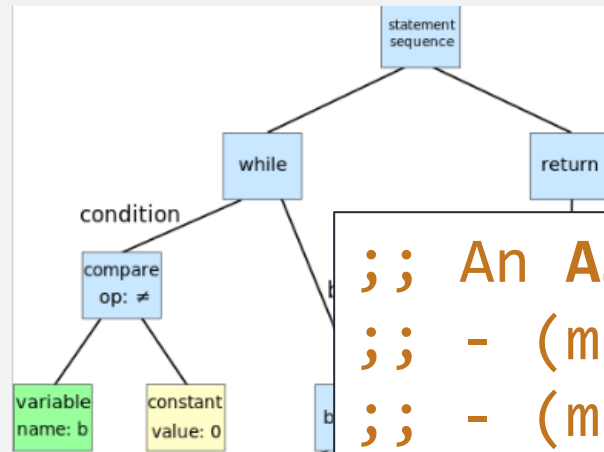


;; An **AST** is one of:  
;; - (mk-num Number)  
;; - (mk-add AST AST)  
;; - (mk-mul AST AST)  
(struct AST [])  
(struct num AST [val])  
(struct add AST [lft rgt])  
(struct mul AST [lft rgt])

Program that does this is called  
**Parser**  
(compiler's first step)

**Program**  
(Surface Syntax)

**Abstract Syntax  
(Program) Tree (AST)**




;; An **AST** is one of:  
;; - (mk-num Number)  
;; - (mk-add AST AST)  
;; - (mk-mul AST AST)  
(struct AST [])  
(struct num AST [val])  
(struct add AST [lft rgt])  
(struct mul AST [lft rgt])

```
;; A Program (Ssexpr) is a:  
;; - Number  
;; - `(Σ ,Program ,Program)  
;; - `(Π ,Program ,Program)
```

```
;; parse: Program -> AST  
;; Converts a Program to an AST
```

```
(define (parse p)  
  (match p  
    [(? number?) ... ] TEMPLATE  
    [`(Σ ,x ,y)  
     ... (parse x) ... (parse y) ... ]  
    [`(Π ,x ,y)  
     ... (parse x) ... (parse y) ... ]))
```



```
;; An AST is one of:  
;; - (mk-num Number)  
;; - (mk-add AST AST)  
;; - (mk-mul AST AST)  
(struct AST [])  
(struct num AST [val])  
(struct add AST [lft rgt])  
(struct mul AST [lft rgt])
```

```
;; A Program (Ssexpr) is a:  
;; - Number  
;; - `(Σ ,Program ,Program)  
;; - `(Π ,Program ,Program)
```

```
;; parse: Program -> AST  
;; Converts a Program to an AST
```

```
(define (parse p)  
  (match p  
    [(? number?) (mk-num p)]  
    [ `(Σ ,x ,y)  
      ... (parse x) ... (parse y) ... ]  
    [ `(Π ,x ,y)  
      ... (parse x) ... (parse y) ... ]))
```

```
;; An AST is one of:  
;; - (mk-num Number)  
;; - (mk-add AST AST)  
;; - (mk-mul AST AST)  
(struct AST [])  
(struct num AST [val])  
(struct add AST [lft rgt])  
(struct mul AST [lft rgt])
```

```
;; A Program (Ssexpr) is a:  
;; - Number  
;; - `(Σ ,Program ,Program)  
;; - `(Π ,Program ,Program)
```

```
;; parse: Program -> AST  
;; Converts a Program to an AST
```

```
(define (parse p)  
  (match p  
    [(? number?) (mk-num p)]  
    [`(Σ ,x ,y)  
     (mk-add (parse x) (parse y))]  
    [`(Π ,x ,y)  
     ... (parse x) ... (parse y) ... ]))
```

```
;; An AST is one of:  
;; - (mk-num Number)  
;; - (mk-add AST AST)  
;; - (mk-mul AST AST)  
(struct AST [])  
(struct num AST [val])  
(struct add AST [lft rgt])  
(struct mul AST [lft rgt])
```

```
;; A Program (Ssexpr) is a:  
;; - Number  
;; - `(Σ ,Program ,Program)  
;; - `(Π ,Program ,Program)
```

```
;; parse: Program -> AST  
;; Converts a Program to an AST
```

```
(define (parse p)  
  (match p  
    [(? number?) (mk-num p)]  
    [`(Σ ,x ,y)  
     (mk-add (parse x) (parse y))]  
    [`(Π ,x ,y)  
     (mk-mul (parse x) (parse y))]))
```

```
;; An AST is one of:  
;; - (mk-num Number)  
;; - (mk-add AST AST)  
;; - (mk-mul AST AST)  
(struct AST [])  
(struct num AST [val])  
(struct add AST [lft rgt])  
(struct mul AST [lft rgt])
```

```
;; A Program (Ssexpr) is a:  
;; - Number  
;; - `(Σ ,Program ,Program)  
;; - `(Π ,Program ,Program)
```

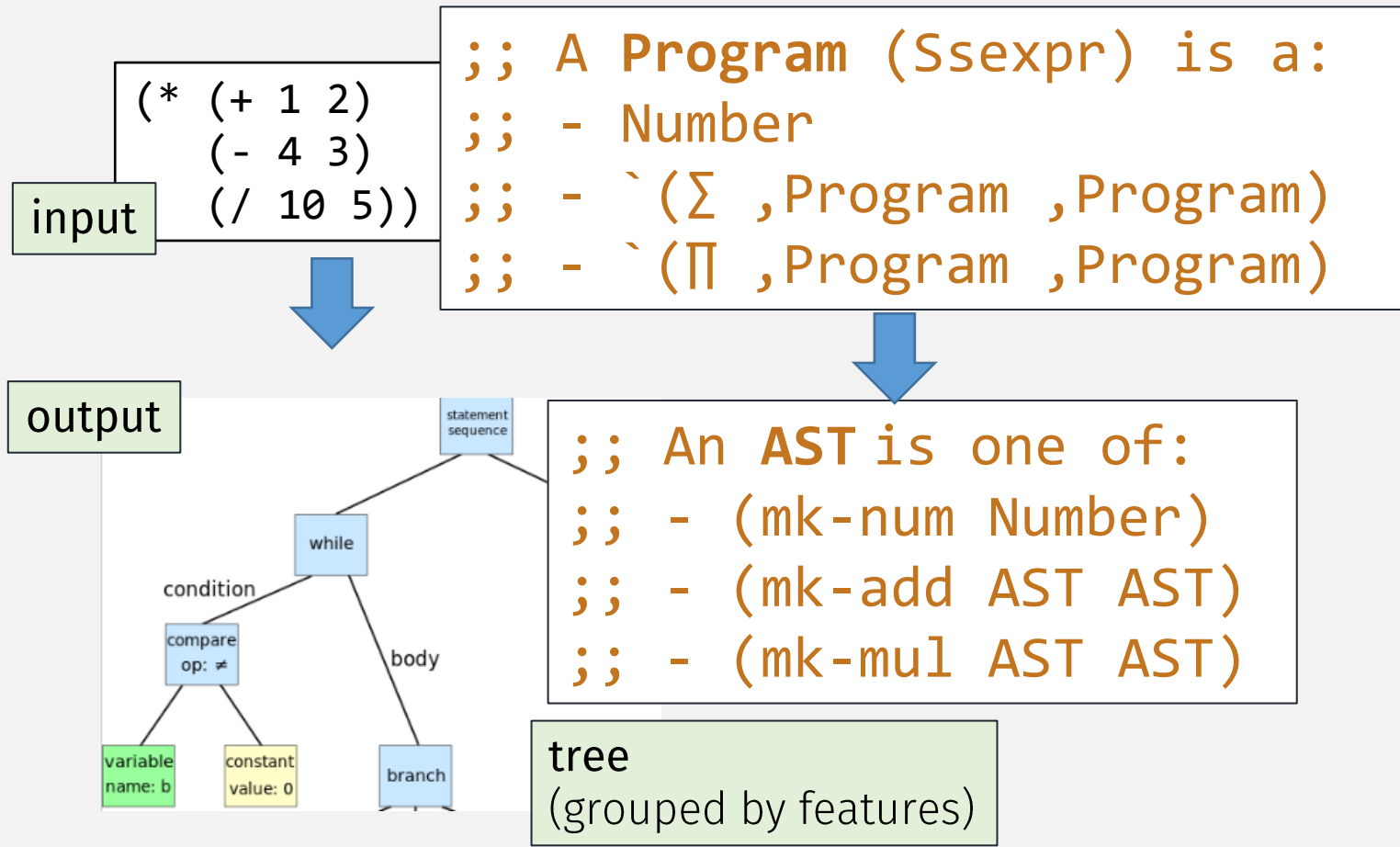
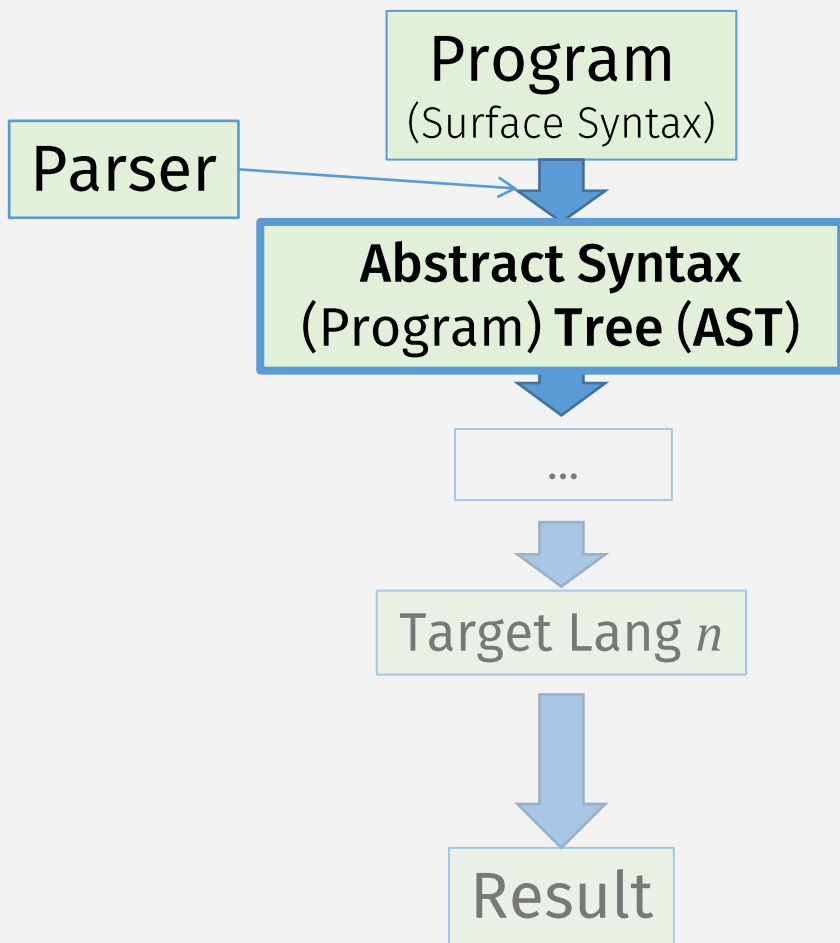
```
;; parse: Program -> AST  
;; Converts a Program to an AST
```

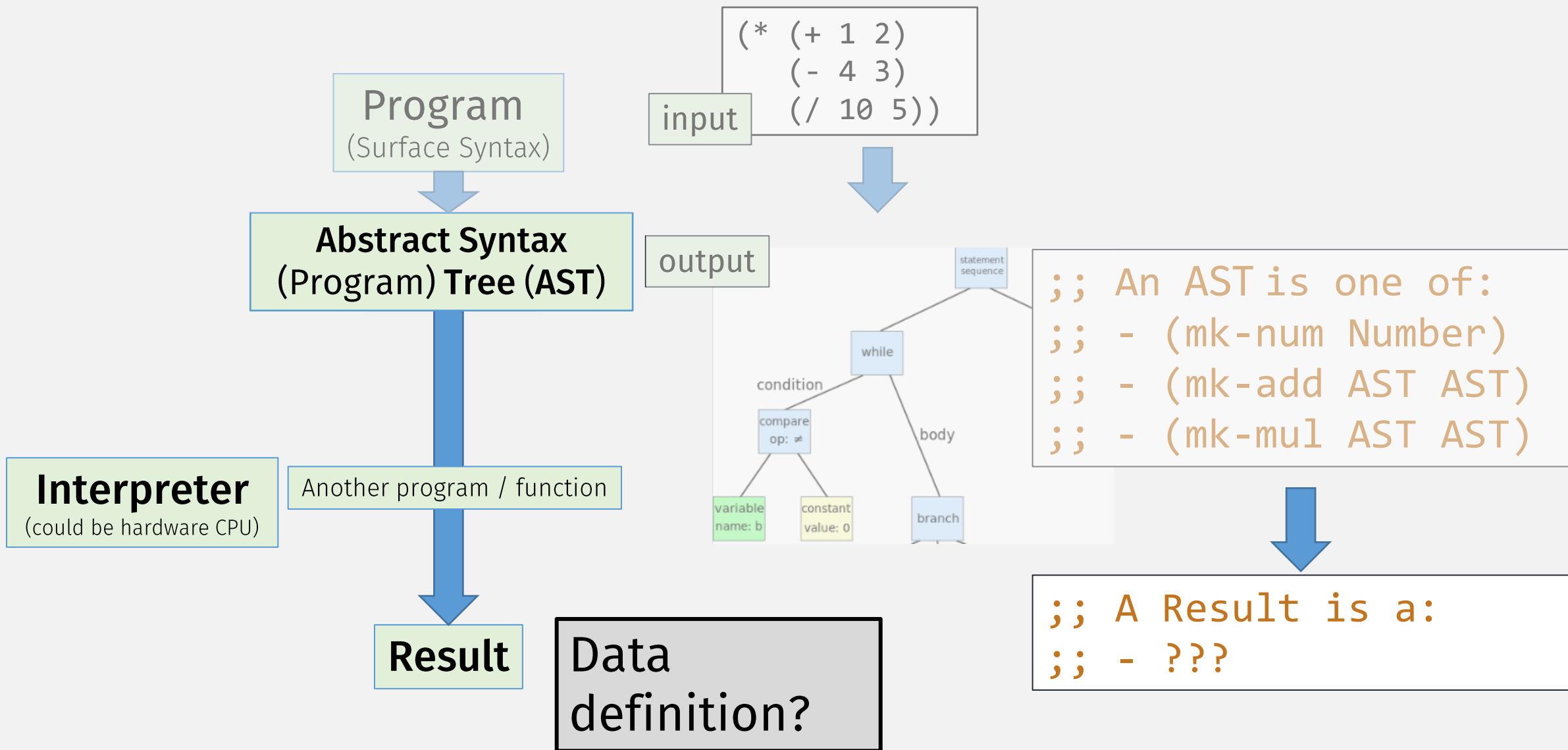
```
(define (parse p)  
  (match p  
    [(? number?) (mk-num p)]  
    [`(Σ ,x ,y)  
     (mk-add (parse x) (parse y))]  
    [`(Π ,x ,y)  
     (mk-mul (parse x) (parse y))]))
```

TEMPLATE MAKES THIS EASY!

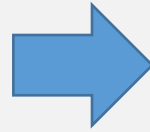
```
;; An AST is one of:  
;; - (mk-num Number)  
;; - (mk-add AST AST)  
;; - (mk-mul AST AST)  
(struct AST [])  
(struct num AST [val])  
(struct add AST [lft rgt])  
(struct mul AST [lft rgt])
```

Previously





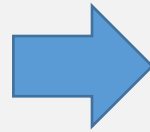
```
;; An AST is one of:  
;; - (mk-num Number)  
;; - (mk-add AST AST)  
;; - (mk-mul AST AST)
```



```
;; A Result is a:  
;; - Number
```

```
;; run: AST -> Result  
;; Computes the Result of running the given program AST
```

```
;; An AST is one of:  
(struct num [val])  
(struct add [lft rgt])  
(struct mul [lft rgt])
```

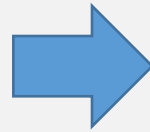


```
;; A Result is a:  
;; - Number
```

```
;; run: AST -> Result  
;; Computes the Result of running the given program AST
```

```
(define (run p) TEMPLATE  
  (match p  
    [(num n) ... ]  
    [(add x y) ... (run x) ...  
     ... (run y) ... ]  
    [(mul x y) ... (run x) ...  
     ... (run y) ... ]))
```

```
;; An AST is one of:  
(struct num [val])  
(struct add [lft rgt])  
(struct mul [lft rgt])
```



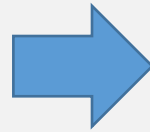
```
;; A Result is a:  
;; - Number
```

```
;; run: AST -> Result  
;; Computes the Result of running the given program AST
```

```
(define (run p)  
  (match p  
    [(num n) n]  
    [(add x y) ... (run x) ...  
                ... (run y) ... ]  
    [(mul x y) ... (run x) ...  
                ... (run y) ... ]))
```

How to combine Results?

```
;; An AST is one of:  
(struct num [val])  
(struct add [lft rgt])  
(struct mul [lft rgt])
```



```
;; A Result is a:  
;; - Number
```

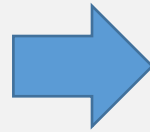
```
;; run: AST -> Result  
;; Computes the Result of running the given program AST
```

```
(define (run p)  
  (match p  
    [(num n) n]  
    [(add x y) (+ (run x)  
                  (run y))]  
    [(mul x y) ... (run x) ...  
                 ... (run y) ... ]])
```

Racket + gives meaning to "Σ" in our new language

How to combine?

```
;; An AST is one of:  
(struct num [val])  
(struct add [lft rgt])  
(struct mul [lft rgt])
```



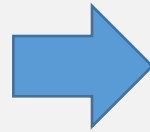
```
;; A Result is a:  
;; - Number
```

```
;; run: AST -> Result  
;; Computes the Result of running the given program AST
```

```
(define (run p)  
  (match p  
    [(num n) n]  
    [(add x y) (+ (run x)  
                  (run y))]  
    [(mul x y) (* (run x)  
                  (run y))])
```

Racket \* gives  
meaning to "Π" in our  
new language

```
;; An AST is one of:  
(struct num [val])  
(struct add [lft rgt])  
(struct mul [lft rgt])
```



```
;; A Result is a:  
;; - Number
```

```
;; run: AST -> Result  
;; Computes the Result of running the given program AST
```

```
(define (run p) TEMPLATE MAKES THIS EASY!  
  (match p  
    [(num n) n]  
    [(add x y) (+ (run x)  
                  (run y))]  
    [(mul x y) (* (run x)  
                  (run y))]))
```

# The “CS 450 LANG” Programming Language

cs450lang.rkt

```
;; parse : Program -> AST
(define (parse p) ...)
;; run: AST -> Result
(define (run t) ...)

(define (eval450 p) (compose run parse))

;; tells Racket to use eval450 when “running” code
;; also, implicitly inserts quote “constructor”
(define-macro (module expr ...)
  (eval450 'expr) ...))
```

`((compose f g) x) = (f (g x))`

cs450lang-prog.rkt

`#lang s-exp “cs450lang.rkt”`

`(Π 3 4) ; => 12`

A program! written  
in “CS450 LANG”!

# “CS450 Lang” Demo

```
#lang s-exp “cs450lang.rkt”
```

```
( $\Pi$  3 4) ; => 12
```

A program! written  
in “CS450 LANG”!

# The “CS450 LANG + BOOLEANS” PL

```
;; A Program is a:  
;; - Number  
;; - `(Σ ,Program ,Program)  
;; - `(Π ,Program ,Program)
```

Both represent code!  
(before it's “run”)

```
;; An AST is one of:  
;; - (mk-num Number)  
;; - (mk-add AST AST)  
;; - (mk-mul AST AST)  
(struct num [val])
```

```
;; An AST is one of  
;; - (mk-num Number)  
;; - (mk-boo Boolean)  
;; - (mk-add AST AST)  
;; - (mk-mul AST AST)  
(struct num [val])  
(struct boo [val])  
(struct add [lft rgt])  
(struct mul [lft rgt])
```

RACKET  
(bool  
value

compiler internal  
(tree) representation  
of the program code

Code that the  
programmer writes

```
;; A Program is a:  
;; - Number  
;; - 450Bool  
;; - `(Σ ,Program ,Program)  
;; - `(Π ,Program ,Program)  
;; A 450Bool is either:  
;; - '450true  
;; - '450false
```

# Parsing “CS450 LANG + BOOLS” Programs

```
;; parse: Program -> AST
;; Converts a “450Lang” S-expression Program to AST
```

```
(define (parse p)
  (match p
    [(? number?) (mk-num p)]
    [(? 450bool?) (mk-boo ??? p)]
    [ `(Σ ,x ,y) (mk-add (parse x) (parse y))]
    [ `(Π ,x ,y) (mk-mul (parse x) (parse y))]))
```

```
;; A Program is a:
;; - Number
;; - 450Bool
;; - `(Σ ,Program ,Program)
;; - `(Π ,Program ,Program)
```

An AST is one of:

- (mk-num Number)
- (mk-boo Boolean)

```
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct num [val])
(struct boo [val])
(struct add [lft rgt])
(struct mul [lft rgt])
```

# Parsing “CS450 LANG + BOOLS” Programs

```
;; parse: Program -> AST  
;; Converts a “450Lang” S-expression Program to AST
```

```
(define (parse p)  
  (match p  
    [(? number?) (mk-num p)] (could you write this function? Just follow the template!)  
    [(? 450bool?) (mk-boo (450bool->bool p))]  
    [ `(Σ ,x ,y) (mk-add (parse x) (parse y)) ]  
    [ `(Π ,x ,y) (mk-mul (parse x) (parse y)) ]))
```

An AST is one of:

- (mk-num Number)
- (mk-boo Boolean)

```
;; - (mk-add AST AST)  
;; - (mk-mul AST AST)  
(struct num [val])  
(struct boo [val])  
(struct add [lft rgt])  
(struct mul [lft rgt])
```

```
;; A Program is a:  
;; - Number  
;; - 450Bool  
;; - `(Σ ,Program ,Program)  
;; - `(Π ,Program ,Program)
```

# Running “CS450 Lang + Bools” Programs

```
;; run: AST -> Result  
;; computes the result of given program AST
```

```
;; A Result is a:  
;; - Number  
;; - Boolean
```

RACKET  
values

```
;; An AST is one of:  
;; ...  
;; - (mk-boo Boolean)  
(struct num [val])  
(struct boo [val])  
(struct add [lft rgt])  
(struct mul [lft rgt])
```

```
(define (run p)  
  (match p  
    [(num n) n]  
    [(boo b) b]  
    [(add x y) (???? (run x) (run y))]  
    [(mul x y) (???? (run x) (run y))]))
```

# Running “CS450 Lang + Bools” Programs

```
;; run: AST -> Result  
;; computes the result of given program AST
```

```
;; A Result is a:  
;; - Number  
;; - Boolean
```

What should happen when two bools are added???

e.g., What is the “meaning” of  $(\Sigma 450\text{true } 450\text{false})$

```
(define (run p)  
  (match p  
    [(num n) n]  
    [(boo b) b]  
    [(add x y) (???? (run x) (run y))]  
    [(mul x y) (???? (run x) (run y))]))
```

# Running “CS450 Lang + Bools” Programs

```
;; run: AST -> Result  
;; computes the result of given program AST
```

```
;; A Result is a:  
;; - Number  
;; - Boolean
```

What should happen when two bools are added???

e.g., What is the “meaning” of `(Σ 450true 450false)`

```
(define (run p)  
  (match p  
    [(num n) n]  
    [(boo b) b]  
    [(add x y) (450+ (run x) (run y))]  
    [(mul x y) (???? (run x) (run y))]))
```

# Running: “CS450LANG” Programs: “450+”

```
;; 450+: Result Result -> Result  
;; “adds” two CS450Lang Result values together
```

```
;; A Result is either:  
;; - Number  
;; - Boolean
```

TEMPLATE

```
(define (450+ x y)  
  (cond  
    [(number? x) ... ]  
    [(boolean? x) ... ])))
```

or

```
(define (450+ x y)  
  (cond  
    [(number? y) ... ]  
    [(boolean? y) ... ])))
```

# Two-Argument Templates

- Sometimes ... a fn must **process two arguments simultaneously**
- **This template should combine templates of both args**
  - (This is only possible if the data defs are simple enough)

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
```

(2-argument) TEMPLATE

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) ... ]
    [(and (number? x) (boolean? y)) ... ]
    [(and (boolean? x) (number? y)) ... ]
    [(and (boolean? x) (boolean? y)) ... ])))
```

(see why this is typically not recommended?)

```
;; 450+: Result Result -> Result  
;; “adds” two CS450Lang Result values together
```

```
(define (450+ x y)  
  (cond  
    [(and (number? x) (number? y)) ... ]  
    [(and (number? x) (boolean? y)) ... ]  
    [(and (boolean? x) (number? y)) ... ]  
    [(and (boolean? x) (boolean? y)) ... ]))
```

```
;; 450+: Result Result -> Result  
;; “adds” two CS450Lang Result values together
```

```
(define (450+ x y)  
  (cond  
    [(and (number? x) (number? y)) (+ x y)]  
    [(and (number? x) (boolean? y)) ... ]  
    [(and (boolean? x) (number? y)) ... ]  
    [(and (boolean? x) (boolean? y)) ... ]))
```

Let's look at other languages!

```
;; 450+: Result Result -> Result  
;; “adds” two CS450Lang Result values together
```

```
(define (450+ x y)  
  (cond  
    [(and (number? x) (number? y)) (+ x y)]  
    [(and (number? x) (boolean? y)) ... ]  
    [(and (boolean? x) (number? y)) ... ]  
    [(and (boolean? x) (boolean? y)) ???]))
```

# “Adding” Booleans

- JAVASCRIPT
  - [repljs.com](https://repljs.com)

When you use the addition operator (+) on boolean values, JavaScript treats `true` as 1 and `false` as 0.

- `true + true = 2`
- `true + false = 1`
- `false + false = 0`

- PYTHON
  - NumPy

When you use the + operator or `np.sum()` on boolean arrays, NumPy treats `True` as 1 and `False` as 0. This is the standard way to count how many elements meet a certain condition.

- **Count all `True` values:** Use [`numpy.sum\(\)`](#) to get the total count.

python

```
import numpy as np
arr = np.array([True, False, True])
total = np.sum(arr) # Returns 2
```

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js/python semantics!)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y)) ... ]
    [(and (boolean? x) (number? y)) ... ]
    [(and (boolean? x) (boolean? y)) ???]))
```

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js/python semantics!)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y)) ... ]
    [(and (boolean? x) (number? y)) ... ]
    [(and (boolean? x) (boolean? y)) (+ (boo->num x) (boo->num y))]))
```

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js/python semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y)) ??? ]
    [(and (boolean? x) (number? y)) ... ]
    [(and (boolean? x) (boolean? y)) (+ (boo->num x) (boo->num y))]))
```

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js/python semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y)) (+ x (boo->num y))]
    [(and (boolean? x) (number? y)) ... ]
    [(and (boolean? x) (boolean? y)) (+ (boo->num x) (boo->num y))]))
```

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js/python semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y)) (+ x (boo->num y))]
    [(and (boolean? x) (number? y)) (+ (boo->num x) y)]
    [(and (boolean? x) (boolean? y)) (+ (boo->num x) (boo->num y))]))
```

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js/python semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y)) (+ x (boo->num y))]
    [(and (boolean? x) (number? y)) (+ (boo->num x) y)]
    [(and (boolean? x) (boolean? y)) (+ (boo->num x) (boo->num y))]))
```

(can any cond clauses be combined?)

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js/python semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ (res->num x) (res->num y))]
    [(and (number? x) (boolean? y)) (+ (res->num x) (res->num y))]
    [(and (boolean? x) (number? y)) (+ (res->num x) (res->num y))]
    [(and (boolean? x) (boolean? y)) (+ (res->num x) (res->num y))]))
```

(can any cond clauses be combined?)

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js/python semantics)
```

```
(define (450+ x y)
  (+ (res->num x) (res->num y)))
```


```
;; A Result is a:
;; - Number
;; - Boolean
```

```
;; res->num: Result -> Number
(define (res->num x)
  (cond
    [(number? x) ...]
    [(boolean? x) ... ]))
```

TEMPLATE

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js/python semantics)
```

```
(define (450+ x y)
  (+ (res->num x) (res->num y)))
```



```
;; res->num: Result -> Number
(define (res->num x)
  (cond
    [(number? x) x]
    [(boolean? x) (boo->num x)]))
```

Can we do the same thing for multiplication?

- Check repljs.com

```
;; 450*: Result Result -> Result  
;; “multiplies” two 450Lang Result values  
;; (following js/python semantics)
```

```
(define (450* x y)  
  (* (res->num x) (res->num y)))
```

```
;; res->num: Result -> Number  
(define (res->num x)  
  (cond  
    [(number? x) x]  
    [(boolean? x) (boo->num x)]))
```

# In-class Coding 4/2: put it all together!

```
;; parse: Program -> AST
;; Parses "450 Lang" Program to AST
```

```
;; run: AST -> Result
;; Computes Result of running a program
```

```
;; An Program is one of:
;; - Number
;; - 450Bool
;; - `(Σ ,Program ,Program)
;; - `(Π ,Program ,Program)
```

Code that the programmer writes

```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-boo Boolean)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct num [val])
(struct boo [val])
(struct add [lft rgt])
(struct mul [lft rgt])
```

Compiler internal  
code representation  
(before it's run)

Use super struct?

```
;; A 450Bool is either
;; - '450true
;; - '450false
```

```
;; A Result is one of:
;; - Number
;; - Boolean
```

RACKET values

Result after  
running the code