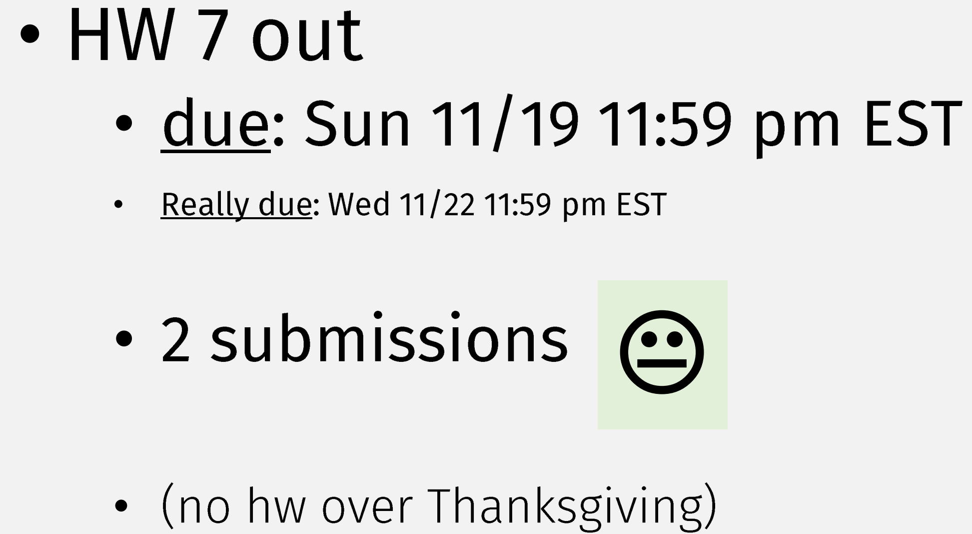
UMass Boston Computer Science

CS450 High Level Languages (section 2)

Defining New Variables in CS450js lang

Monday, November 20, 2023

Logistics



- HW 7 out
  - <u>due</u>: Sun 11/19 11:59 pm EST
    - <u>Really due</u>: Wed 11/22 11:59 pm EST
  - 2 submissions 😐
  - (no hw over Thanksgiving)

HW 5 (Pong Game) recap

Key Points

- Think about data organization
  - As `World` gets more components, need subgroupings of data
  - Be careful with: **extraneous data**, e.g., **does player need x data?**

```
;; A WorldState is a:
;; (world x y vx vy p1 p2 sc1 sc2)
;; x   : XCoordinate
;; y   : YCoordinate
;; vx  : Velocity
;; vy  : Velocity
;; p1  : YCoordinate
;; p2  : YCoordinate
;; sc1 : NaturalNum in [0,
;; sc2 : NaturalNum in [0,
```

or

```
;; A WorldState is a
;; (world b p1 p2)
;; b : Ball
;; p1 : Player
;; p2 : Player
```

vs

```
;; A WorldState is a
;; (world b p1 p2 sc1 sc2)
;; b   : Ball
;; p1  : YCoordinate
;; p2  : YCoordinate
;; sc1 : NatNum in [0,10]
;; sc2 : NatNum in [0,10]
```

HW 5 (Pong Game) recap

Key Points

- Think about data organization
- Data organization affects code organization (and readability)

```
;; next-world : WorldState -> WorldState
(define (next-world w)
  (match-define (world b p1 p2) w)
  (world ... Update ball ...
         ... Update player1 ...
         ... Update player2 ...
```

```
;; A WorldState is a
;; (world b p1 p2)
;; b : Ball
;; p1 : Player
;; p2 : Player
```

HW 5 (Pong Game) recap

Key Points

- Think about data organization
- Data organization affects code organization (and **readability**)

Think about fn names!
"ball-in-scene/x?" no longer makes sense!!!

```
;; next-world : WorldState -> WorldState
(define (next-world w)
  (match-define (world b p1 p2) w)
  (world (if (scored? b)
             (mk-center-ball/rand-val)
             (next-ball b p1 p2))
         ... Update player1 ...
         ... Update player2 ...
```

```
;; A WorldState is a
;; (world b p1 p2)
;; b : Ball
;; p1 : Player
;; p2 : Player
```

next-world

vs

```
    (if (> mouse-y (- SCENE-HEIGHT PADDLE-HEIGHT))
        (- SCENE-HEIGHT PADDLE-HEIGHT)
        mouse-y)))

  (define ball-x-next (+ (world-x w) (world-xvel w)))
  (define ball-y-next (+ (world-y w) (world-yvel w)))

  ;; Check for collision with the paddles
  (define paddle1-collision?
    (and (<= (world-x w) (+ PADDLE-PLAYER1-INIT-X PADDLE-WIDTH))
         (<= (world-y w) (+ (world-player1-y w) PADDLE-HEIGHT))
         (>= (world-y w) (world-player1-y w))))

  (define paddle2-collision?
    (and (>= (world-x w) (- PADDLE-PLAYER2-INIT-X BALL-SIZE))
         (<= (world-y w) (+ (world-player2-y w) PADDLE-HEIGHT))
         (>= (world-y w) (world-player2-y w))))

  ;; If a collision with a paddle occurs, reverse the ball's horizontal velocity
  (define ball-x-vel-bounce
    (if (or paddle1-collision? paddle2-collision?)
        (- (world-xvel w))
        (world-xvel w)))

  (define ball-y-vel-next
    (if (or (<= (sub1 ball-y-next) 0) (>= (add1 ball-y-next) SCENE-HEIGHT))
        (- (world-yvel w))
        (world-yvel w)))

  ;; Check for scoring
  (define player1-scores? (>= (world-x w) SCENE-WIDTH))
  (define player2-scores? (<= (world-x w) 0))

  (define player1-score
    (if player1-scores? (+ 1 (world-player1-score w)) (world-player1-score w)))

  (define player2-score
    (if player2-scores? (+ 1 (world-player2-score w)) (world-player2-score w)))

  (define new-x
    (if (or player1-scores? player2-scores?)
        (/ SCENE-WIDTH 2)
        (+ (world-x w) ball-x-vel-bounce)))

  (define new-y
    (if (or player1-scores? player2-scores?)
        (random (- SCENE-HEIGHT BALL-SIZE))
        (+ (world-y w) ball-y-vel-next)))

  (define new-xvel
    (if (or player1-scores? player2-scores?)
        (if player1-scores? BALL-SPEED (- BALL-SPEED))
        ball-x-vel-bounce))

  (define new-yvel
    (if (or player1-scores? player2-scores?)
        (random BALL-SPEED)
        ball-y-vel-next))

  (make-world
   new-x
   new-y
   new-xvel
   new-yvel
   (world-player1-y w)
   player2-y-next
   player1-score
   player2-score))
```

Which is more readable (by humans)?

60 lines

```
;; next-world : WorldState -> WorldState
;; Computes the next world state from the given one
(define/contract (next-world w)
  (-> world? world?)
  (match-define (world x y xvel yvel left-paddle-y right-paddle-y left-score right-score winner) w)
  (define (reset? x y)
  (or (or (< x (+ LEFT-EDGE BALL-RADIUS))
          (> x (- RIGHT-EDGE BALL-RADIUS)))
      (and (= x 0) (= y 0))))

  (define new-left-score left-score)
  (define new-right-score right-score)
  (define new-winner winner)


  (define (left-paddle-collision? x y)
    (and (<= x (+ LEFT-PADDLE-X BALL-RADIUS))
         (>= y (- left-paddle-y (/ PADDLE-HEIGHT 2)))
         (<= y (+ left-paddle-y (/ PADDLE-HEIGHT 2)))))

  (define (right-paddle-collision? x y)
    (and (>= x (- RIGHT-PADDLE-X BALL-RADIUS))
         (>= y (- right-paddle-y (/ PADDLE-HEIGHT 2)))
         (<= y (+ right-paddle-y (/ PADDLE-HEIGHT 2)))))

  (define new-xvel
    (cond
      [(reset? x y) (random-velocity)]
      [(left-paddle-collision? x y) (abs xvel)]
      [(right-paddle-collision? x y) (- (abs xvel))]
      [else (if (ball-in-scene/x? (make-x-coordinate x)) xvel (- xvel))]))


  (define new-yvel
    (cond
      [(reset? x y) (random-velocity)]
      [(ball-in-scene/y? (make-y-coordinate y)) yvel]
      [else (- yvel)]))

  (define new-x
    (if (reset? x y)
        (begin
          (cond
            [(< x (+ LEFT-EDGE BALL-RADIUS))
                 new-right-score (+ 1 right-score))
               = new-right-score 10)
              egin
               (set! new-winner "Right")
               (display "Player Right wins!\n"))
              et! new-winner ""))
            HT-EDGE 2)]

             - RIGHT-EDGE BALL-RADIUS))
               new-left-score (+ 1 left-score))
               = new-left-score 10)
              (begin
                (set! new-winner "Left")
                (display "Player Left wins!\n"))
              (set! new-winner ""))
            (/ RIGHT-EDGE 2)]
           [else
            (/ RIGHT-EDGE 2)]))
        (+ x new-xvel)))

  (define new-y
    (if (reset? x y)
        (/ BOTTOM-EDGE 2)
        (+ y new-yvel)))
  ; Check if the game is over and display the winning message
  (if (or (= new-left-score 10) (= new-right-score 10))
      (make-world new-x new-y new-xvel new-yvel left-paddle-y right-paddle-y new-left-score new-right-score new-winner)
; If the game is not over, continue updating the world
      (make-world new-x new-y new-xvel new-yvel left-paddle-y right-paddle-y new-left-score new-right-score winner)))
```

70 lines

Don't submit this kind of code in this class (or in software engineering)

```
;; Computes the next world state from the given one
(define/contract (next-world w)
  (-> world? world?)
  (match-define (world x y xvel yvel left-paddle-y right-paddle-y left-score right-score winner) w)
  (define (reset? x)
    (or (< x (+ LEFT-EDGE BALL-RADIUS))
        (> x (- RIGHT-EDGE BALL-RADIUS))
        (= x 0)))

  (define new-left-score left-score)
  (define new-right-score right-score)

  (define (left-paddle-collision? x y)
    (and (<= x (+ LEFT-PADDLE-X BALL-RADIUS))
         (>= y (- left-paddle-y (/ PADDLE-HEIGHT 2)))
         (<= y (+ left-paddle-y (/ PADDLE-HEIGHT 2)))))

  (define (right-paddle-collision? x y)
    (and (>= x (- RIGHT-PADDLE-X BALL-RADIUS))
         (>= y (- right-paddle-y (/ PADDLE-HEIGHT 2)))
         (<= y (+ right-paddle-y (/ PADDLE-HEIGHT 2)))))

  (define new-xvel
    (cond
      [(reset? x) (random-velocity)]
      [(left-paddle-collision? x y) (abs xvel)]
      [(right-paddle-collision? x y) (- (abs xvel))]
      [else (if (ball-in-scene/x? (make-x-coordinate x)) xvel (- xvel))]))


  (define new-yvel
    (cond
      [(reset? x) (random-velocity)]
      [(ball-in-scene/y? (make-y-coordinate y)) yvel]
      [else (- yvel)]))

  (define new-x
    (if (reset? x)
        (begin
          (cond
            [(< x (+ LEFT-EDGE BALL-RADIUS))
             (set! new-right-score (+ 1 right-score))
             (if (= new-right-score 10)
                 (begin
                   (set! winner "Right")
                   (display "Player Right wins!\n"))
                 (set! winner ""))
             (/ RIGHT-EDGE 2)]

            [(> x (- RIGHT-EDGE BALL-RADIUS))
             (set! new-left-score (+ 1 left-score))
             (if (= new-left-score 10)
                 (begin
                   (set! winner "Left")
                   (display "Player Left wins!\n"))
                 (set! winner ""))
             (/ RIGHT-EDGE 2)]
            [else
             (/ RIGHT-EDGE 2)]))
        (+ x new-xvel)))

  (define new-y
    (if (reset? x)
        (/ BOTTOM-EDGE 2)
        (+ y new-yvel)))

  ;; Shows us who won in console mode
  (cond
    [(= new-left-score 10)
     (begin
       (set! winner "Left")
       (display "Player Left wins!\n"))]
    [(= new-right-score 10)
     (begin
       (set! winner "Right")
       (display "Player Right wins!\n"))]
    [else '()])
  (make-world new-x new-y new-xvel new-yvel left-paddle-y right-paddle-y new-left-score new-right-score winner))
```

80 lines

HW 5 (Pong Game) recap

Key Points

- Think about data organization
- Data organization affects code organization (and readability)
- Break up large data defs into (logical) smaller, readable ones
- Break up large functions into (logical) smaller, readable ones

Google C++ Style Guide

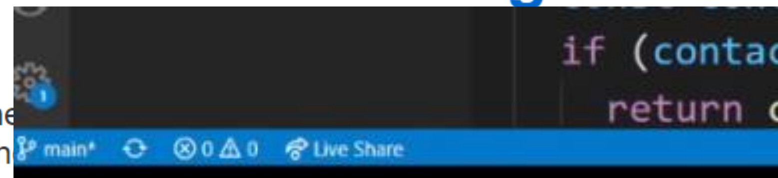
Write Short Functions

Prefer small and focused functions.

We recognize that long functions are some... ions length. If a function exceeds about 40 lines, think about wheth... f the program.

Even if your long function works perfectly ... new be result in bugs that are hard to find. Keepin... other p modify your code. Small functions are also...

You could find long and complicated functi... ated by code: if working with such a function prove..., or you piece of it in several different contexts, cor... manag...



Why I Never Write Long Functions

Web Dev Simplified

1.44M subscribers

Subscribe

100K views 2 years ago Clean Code

Small Functions Considered Awesome

Josh Saint Jacque · Follow

11 min read · Aug 22, 2017

HW 6 (editor + mouse) Recap

Key Points

- Designing functions with accumulators

```
;; mouse-handler : Editor Coord Coord MouseEvt -> Editor
(define (mouse-handler ed x y mevt)
  (cond
    [(mouse=/ mevt "button-down") (split ed x)]
    [else ed]))
```

new pre initial (chars-sofar): empty (no chars)

new post initial (rest-chars): (ed->list ed0) (all chars)

Editor (new)

new pre | new post

Invariant: "so far" + "remaining" = ed0

Need to compute new split (at mouse x)

Editor ed0 (current)

pre | post

HW 6 (editor + mouse) Recap

new `pre` initial (`chars-sofar`): `empty` (no chars)

Key Points

- Designing functions with accumulators

```
(define (split ed0 x) ;; split : Editor Coord -> Editor
  ;; split/a : List<1str> List<1str> Coord -> Editor
  ;; ACCUMULATOR: chars-so-far
  ;; invariant: represents (rev) of chars seen so far, where
  ;;   (append (rev chars-so-far) rest-chars) = (ed->lst ed0)
  ;;       and: (image-width (render-chars chars-sofar)) < x
  (define (split/a chars-sofar rest-chars x)
    (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]
          [else          (match-define (cons c rst) rest-chars)
           (if (chars-at-cursor? x (cons c chars-sofar))
               (editor  (cons c chars-sofar) rst)
               (split/a (cons c chars-sofar) rst x))]))
  (split/a empty (ed->lst ed0) x))
```

HW 6 (editor + mouse) Recap

new **pre** initial (chars-sofar): empty (no chars)

new **post** initial (rest-chars): (ed->list ed0) (all chars)

Key Points

- Designing functions with accumulators

```
(define (split ed0 x) ;; split : Editor Coord -> Editor
  ;; split/a : List<1str> List<1str> Coord -> Editor
  ;; ACCUMULATOR: chars-so-far
  ;; invariant: represents (rev) of chars seen so far, where
  ;;   (append (rev chars-so-far) rest-chars) = (ed->lst ed0)
  ;;       and: (image-width (render-chars chars-sofar)) < x
  (define (split/a chars-sofar rest-chars x)
    (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]
          [else          (match-define (cons c rst) rest-chars)
           (if (chars-at-cursor? x (cons c chars-sofar))
               (editor  (cons c chars-sofar) rst)
               (split/a (cons c chars-sofar) rst x))]))
  (split/a empty (ed->lst ed0) x))
```

HW 6 (editor + mouse) Recap

new **pre** initial (**chars-sofar**): **empty** (no chars)

new **post** initial (**rest-chars**): **(ed->list ed0)** (all chars)

Key Points

- Designing functions with accumulators

```
(define (split ed0 x) ;; split : Editor Coord -> Editor
  ;; split/a : List<1str> List<1str> Coord -> Editor
  ;; ACCUMULATOR: chars-so-far
  ;; invariant: represents (rev) of chars seen so far, where
  ;;   (append (rev chars-so-far) rest-chars) = (ed->lst ed0)
  ;;        and: (image-width (render-chars chars-sofar)) < x
  (define (split/a chars-sofar rest-chars x)
    (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]
          [else          (match-define (cons c rst) rest-chars)
           (if (chars-at-cursor? x (cons c chars-sofar))
               (editor  (cons c chars-sofar) rst)
               (split/a (cons c chars-sofar) rst x))]))
  (split/a empty (ed->lst ed0) x))
```

Invariant:
"so far" +
"remaining"
= ed0

HW 6 (editor + mouse) Recap

Key Points

- Designing functions with accumulators

```
(define (split ed0 x) ;; split : Editor Coord -> Editor
  ;; split/a : List<1str> List<1str> Coord -> Editor
  ;; ACCUMULATOR: chars-so-far
  ;; invariant: represents (rev) of chars seen so far, where
  ;;   (append (rev chars-so-far) rest-chars) = (ed->lst ed0)
  ;;      and: (image-width (render-chars chars-sofar)) < x
  (define (split/a chars-sofar rest-chars x)
    (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]
          [else         (match-define (cons c rst) rest-chars)
           (if (chars-at-cursor? x (cons c chars-sofar))
               (editor  (cons c chars-sofar) rst)
               (split/a (cons c chars-sofar) rst x))]))
  (split/a empty (ed->lst ed0) x))
```

TEMPLATE

????

????

????

HW 6 (editor + mouse) Recap

Key Points

- Designing functions with accumulators

```
(define (split ed0 x) ;; split : Editor Coord -> Editor
  ;; split/a : List<1str> List<1str> Coord -> Editor
  ;; ACCUMULATOR: chars-so-far
  ;; invariant: represents (rev) of chars seen so far, where
  ;;   (append (rev chars-so-far) rest-chars) = (ed->lst ed0)
  ;;       and: (image-width (render-chars chars-sofar)) < x
  (define (split/a chars-sofar rest-chars x)
    (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]
          [else         (match-define (cons c rst) rest-chars)
           (if (chars-at-cursor? x (cons c chars-sofar))
               (editor  (cons ???? chars-sofar) rst)
               (split/a (cons c chars????sofar) rst x))]))
  (split/a empty (ed->lst ed0) x))
```

TEMPLATE

HW 6 (editor + mouse) Recap

Key Points

- Designing functions with accumulators

```
(define (split ed0 x) ;; split : Editor Coord -> Editor
  ;; split/a : List<1str> List<1str> Coord -> Editor
  ;; ACCUMULATOR: chars-so-far
  ;; invariant: represents (rev) of chars seen so far, where
  ;;   (append (rev chars-so-far) rest-chars) = (ed->lst ed0)
  ;;      and: (image-width (render-chars chars-sofar)) < x
  (define (split/a chars-sofar rest-chars x)
    (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]
          [else         (match-define (cons c rst) rest-chars)
           (if (chars-at-cursor? x (cons c chars-sofar))
               (editor  (cons ???? chars-sofar) rst)
               (split/a (cons c chars????sofar) rst x))]))
  (split/a empty (ed->lst ed0) x))
```

Empty case

Invariant:
"so far" +
"remaining"
= ed0 ☑

HW 6 (editor + mouse) Recap

Key Points

- Designing functions with accumulators

```
(define (split ed0 x) ;; split : Editor Coord -> Editor
  ;; split/a : List<1str> List<1str> Coord -> Editor
  ;; ACCUMULATOR: chars-so-far
  ;; invariant: represents (rev) of chars seen so far, where
  ;;   (append (rev chars-so-far) rest-chars) = (ed->lst ed0)
  ;;       and: (image-width (render-chars chars-sofar)) < x
  (define (split/a chars-sofar rest-chars x)
    (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]
          [else          (match-define (cons c rst) rest-chars)
           (if (chars-at-cursor? x (cons c chars-sofar))
               (editor  (cons???? chars-sofar) rst)
               (split/a (cons c chars????sofar) rst x))]))
  (split/a empty (ed->lst ed0) x))
```

Non-empty case?

Need one more **invariant!**

Invariant:
"so far" +
"remaining"
= ed0 ☑

Done!

Keep going ...

HW 6 (editor + mouse) Recap

Key Points

- Designing functions with accumulators

```
(define (split ed0 x) ;; split : Editor Coord -> Editor
  ;; split/a : List<1str> List<1str> Coord -> Editor
  ;; ACCUMULATOR: chars-so-far
  ;; invariant: represents (rev) of chars seen so far, where
  ;;   (append (rev chars-so-far) rest-chars) = (ed->lst ed0)
  ;;        and: (image-width (render-chars chars-sofar)) < x
  (define (split/a chars-sofar rest-chars x)
    (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]
          [else          (match-define (cons c rst) rest-chars)
           (if (chars-at-cursor? x (cons c chars-sofar))
               (compute-before/after x chars-sofar c rst)
               (split/a (cons c chars-sofar) rst x))]))
  (split/a empty (ed->lst ed0) x))
```

Done????

Need to compute whether cursor is before or after!

HW 6 (editor + mouse) Recap

Key Points

- Designing functions with accumulators
- Code should be sufficiently "readable" such that …
    - You can present your code like I just did!
    - (Still on the table for this semester …)

Last Time

Introducing: The “CS450js” Programming Lang!

Programmer writes:

Next Feature: **Variables?**

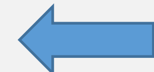
```
;; A 450jsExpr is one of:
;; - Number
;; - String
;; - (list ‘+ 450jsExpr 450jsExpr)
;; - (list ‘- 450jsExpr 450jsExpr)
```

parse450js

```
;; A 450jsAST is one of:
;; - (num Number)
;; - (str String)
;; - (add 450jsAST 450jsAST)
;; - (sub 450jsAST 450jsAST)

(struct num [val])
(struct str [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

“eval450js”



run450js

(JS semantics)

```
;; A 450jsResult is one of:
;; - Number
;; - String
;; - NaN
```

“meaning” of the program

Last Time

NOTE: not needed for hw7

Adding Variables

```
;; A Variable is a Symbol
```

```
;; A 450jsExpr is one of:
;; - Number
;; - String
;; - Variable
;; - (list '+ 450j
;; - (list '- 450jsExpr 450jsExpr)
```

parse450js

```
;; A 450jsAST is one of:
;; - (num Number)
;; - (str String)
;; - (var Symbol)
;; - (add 450jsAST 450jsAST)
;; - (sub 450jsAST 450jsAST)
```

```
(struct num [val])
(struct str [val])
(struct var [name])
(struct add [lft rgt])
```

```
;; A 450jsResult is one of:
;; - Number
;; - String
;; - NaN
```

run450js

Q$_1$: What is the "meaning" of a variable?

A$_1$: Whatever "value" it is bound to

Q$_2$: Where do these "values" come from?

A$_2$: Other parts of the program

The run function needs to "remember" these values (with an **accumulator**!)

run450js, with an accumulator

```
;; run: 450jsAST -> 450jsResult
;; Computes result of running a CS450js program AST
(define (run p)
  ;; accumulator acc : Environment
  ;; invariant: Contains variable+result pairs that are currently in-scope
  (define (run/acc p acc)
    (match p
     [(num n) n]
     [(add x y) (450+ (run/acc x) (run/acc y))]))
 (run/acc p  ???  ))
```

Environments

- A data structure that "associates" two things together
  - E.g., maps, hashes, etc
  - For simplicity, let's use list-of-pairs

```
;; An Environment is one of:
;; - empty
;; - (cons (list Var 450jsResult) Environment)
;; interpretation: a runtime environment for
;; (ie gives meaning to) cs450js-lang variables
;; if there are duplicates,
;; vars at front of list shadow those in back
```

Environments

- A data structure that “associates” two things together
  - E.g., `maps`, hashes, etc
  - For simplicity, let’s use list-of-pairs
- Needed operations:
  - `env-add    : Env Var Result -> Env`
  - `env-lookup : Env Var -> Result`

- Repo: **cs450f23/lecture21-inclass**
- File: **env-**<*your last name*>**.rkt**

In-class Coding 11/21: **write env ops**

- Needed operations:
  - `env-add    : Env Var 450jsResult -> Env`
  - `env-lookup : Env Var -> 450jsResult`

```
;; An Environment (Env) is one of:
;; - empty
;; - (cons (list Var 450jsResult) Environment)
```

```
;; interpretation: a runtime environment for
;; (ie gives meaning to) cs450js-lang variables
```

```
;; if there are duplicates,
;; vars at front of list shadow those in back
```

Think about examples where this happens!

run450js, with an Environment

TODO:
- When are variables "added" to environment
- Initial environment?

```
;; run: 450jsAST -> 450jsResult
;; Computes result of running CS450js AST
```

```
(define (run p)
  ;; accumulator env : Environment
  ;; invariant: Contains variable+result pairs that are in-scope
  (define (run/acc p env)
    (match p
     [(num n) n]
     [(var x) (lookup env x)]
     [(add x y) (450+ (run/acc x env) (run/acc y env))]))
 (run/acc p  ???  ))
```

NOTE: not needed for hw7

Programs that Create Variables

```
;; A 450jsExpr is one of:
;; - Number
;; - String
;; - Variable
;; - (list 'bind Var 450jsExpr 450jsExpr)
;; - (list '+ 450jsExpr 450jsExpr)
;; - (list '- 450jsExpr 450jsExpr)
```

(sometimes called “let”)

```
;; A 450jsAST is one of:
;; - (num Number)
;; - (str String)
;; - (var Symbol)
;; - (bind Symbol 450jsAST 450jsAST)
;; - (add 450jsAST 450jsAST)
;; - (sub 450jsAST 450jsAST)

(struct num [val])
(struct str [val])
(struct var [name])
(struct bind [var expr body])
(struct add [lft rgt])
(struct sub [lft rgt])
```

run450js, with an Environment

```
;; run: 450jsAST -> 450jsResult
(define (run p)
  ;; accumulator env : Environment
  ;; invariant: Contains variable:result pairs that are in scope
  (define (run/e p env)
    (match p
     [(num n) n]
     [(var x) (lookup env x)]
     [(bind x e body) (run/e body (add env x (run/e e env)))]
     [(add x y) (450+ (run/e x env) (run/e y env))]))
 (run/e p  ???  ))
```

3. run body with that new environment

2. add variable x to environment

1. Compute 450jsResult that variable x represents

Initial Environment

TODO:
- ~~When are variables "added" to environment~~
- Initial environment?

```
(define (run p)
  ;; accumulator env : Environment
  ;; invariant: Contains variable+result pairs that are in-scope
  (define (run/e p env)
    (match p
     [(num n) n]
     [(var x) (lookup env x)]
     [(bind x e body) (run/e body (add env x (run/e e env)))]
     [(add x y) (450+ (run/e x env) (run/e y env))]))
 (run/e p  ???  ))
```

Initial Environment

TODO:

- ~~When are variables "added" to environment~~
- Initial environment?

```
;; A 450jsExpr is one of:
;; - Number
;; - String
;; - Variable
;; - (list 'bind Var 450jsExpr 450jsExpr)
;; - (list '+ 450jsExpr 450jsExpr)
;; - (list '- 450jsExpr 450jsExpr)
```

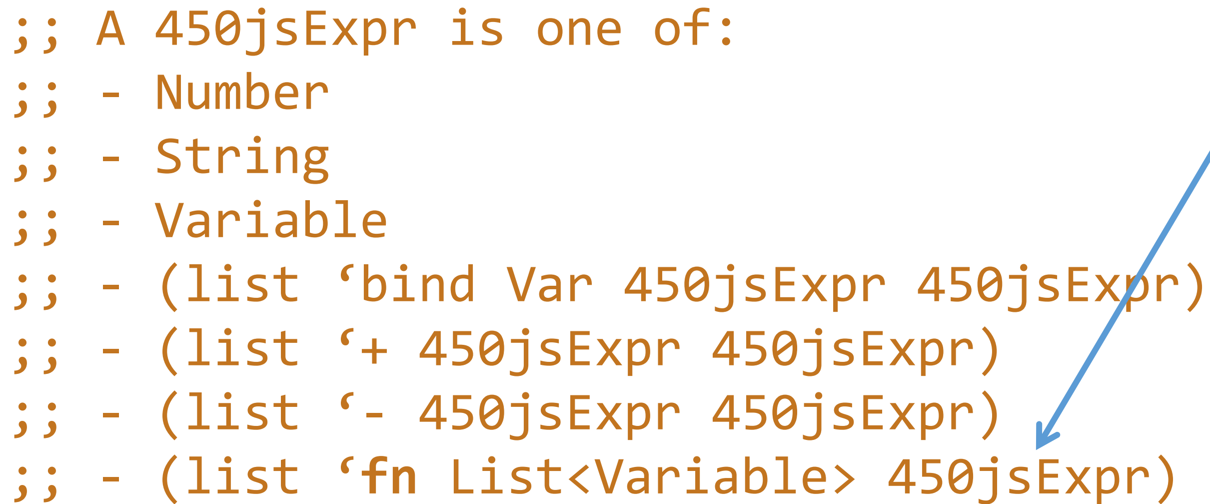
Should these be different language constructs?

Can these go into initial env?

Next Feature: **Functions?**

Initial Environment

TODO:
- ~~When are variables "added" to environment~~
- Initial environment?
- Function representation



Next Feature: **Functions?**

Initial Environment

TODO:
- ~~When are variables "added" to environment~~
- Initial environment?
- Function representation
- Function result?

```
;; A 450jsExpr is one of:
;; - Number
;; - String
;; - Variable
;; - (list 'bind Var 450jsExpr 450jsExpr)
;; - (list '+ 450jsExpr 450jsExpr)
;; - (list '- 450jsExpr 450jsExpr)
;; - (list 'fn List<Variable> 450jsExpr)
```

```
;; A 450jsResult is one of:
;; - Number
;; - String
;; - NaN
;; - ???
```

Initial Environment

TODO:
- ~~When are variables "added" to environment~~
- Initial environment?
- Function representation
- Function result?
- Function calls?

```
;; A 450jsExpr is one of:
;; - Number
;; - String
;; - Variable
;; - (list 'bind Var 450jsExpr 450jsExpr)
;; - (list '+ 450jsExpr 450jsExpr)
;; - (list '- 450jsExpr 450jsExpr)
;; - (list 'fn List<Variable> 450jsExpr)
;; - (list 'fncall 450jsExpr 450jsExpr)
```

```
;; A 450jsResult is one of:
;; - Number
;; - String
;; - NaN
;; - ???
```

Initial Environment

TODO:
- ~~When are variables "added" to environment~~
- Initial environment?
- Function representation
- Function result?
- Function calls?

```
;; A 450jsExpr is one of:
;; - Number
;; - String
;; - Variable
;; - (list 'bind Var 450jsExpr 450jsExpr)
;; - (list '+ 450jsExpr 450jsExpr)
;; - (list '- 450jsExpr 450jsExpr)
;; - (list 'fn List<Variable> 450jsExpr)
;; - (list 'fncall 450jsExpr 450jsExpr)
```

```
;; A 450jsResult is one of:
;; - Number
;; - String
;; - NaN
;; - ???
```

Initial Environment

TODO:
- ~~When are variables "added" to environment~~
- Initial environment?
- Function representation
- Function result?
- Function calls?

```
;; A 450jsExpr is one of:
;; - Number
;; - String
;; - Variable
;; - (list 'bind Var 450jsExpr 450jsExpr)
;; - (list '+ 450jsExpr 450jsExpr)
;; - (list '- 450jsExpr 450jsExpr)
;; - (list 'fn List<Variable> 450jsExpr)
;; - (list 'fncall 450jsExpr 450jsExpr)
```

```
;; A 450jsResult is one of:
;; - Number
;; - String
;; - NaN
;; - ???
```

Most language function calls don't explicitly require saying "function call"

No More Quizzes!

but push your in-class work to:
Repo: **`cs450f23/lecture21-inclass`**