UMass Boston Computer Science

CS450 High Level Languages (section 2)

Scoping

Monday, November 27, 2023

Logistics

- HW 7 in
  - due: ~~Sun 11/19 11:59 pm EST~~
  - Really due: ~~Wed 11/22 11:59 pm EST~~

- HW 8 out
  - due: Sun 12/3 11:59 pm EST

Last Time

The “CS450JS” Programming Lang! (so far)

```
;; A 450jsAtom (Atom) is:
;; - Number
;; - String
;; - …
```

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

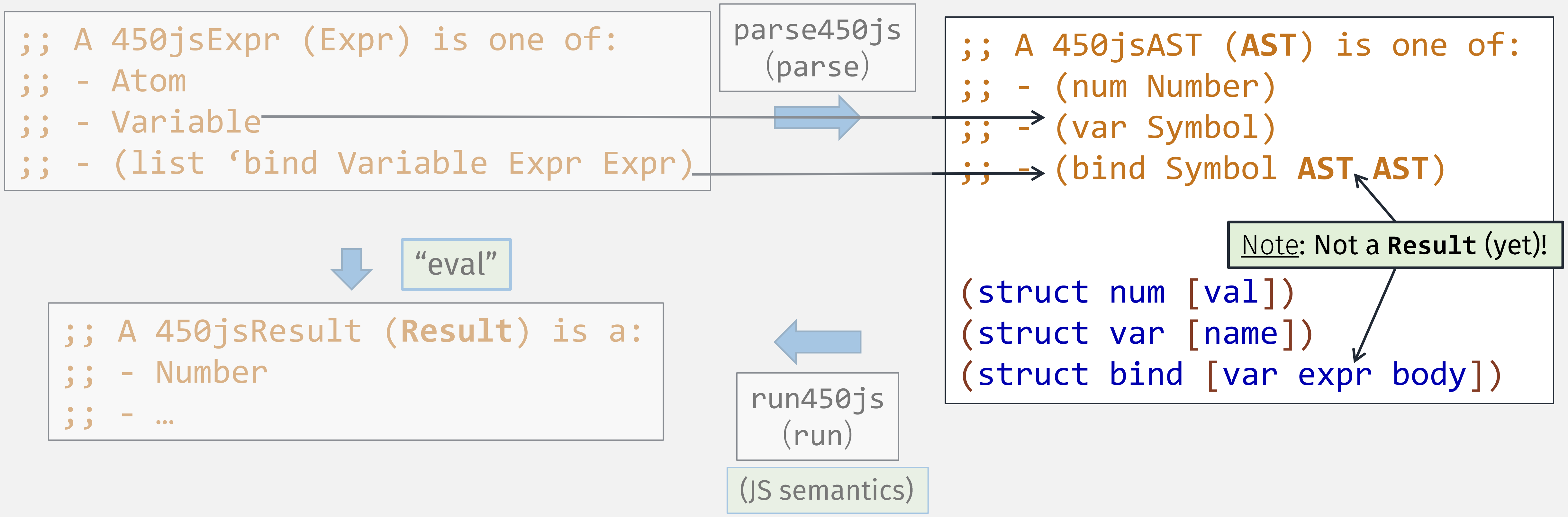
```
;; A 450jsExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - (list ‘bind Variable Expr Expr
```

Variable reference

Create new variables

Last Time

The “CS450JS” Programming Lang! (so far)



```
;; A 450jsExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - (list ‘bind Variable Expr Expr)
```

```
parse450js
(parse)
```

```
;; A 450jsAST (AST) is one of:
;; - (num Number)
;; - (var Symbol)
;; - (bind Symbol AST AST)
```

Note: Not a **Result** (yet)!

```
(struct num [val])
(struct var [name])
(struct bind [var expr body])
```

“eval”

```
;; A 450jsResult (Result) is a:
;; - Number
;; - …
```

```
run450js
(run)
```

(JS semantics)

run450js (with an accumulator)

Environment has **Results** (not **AST**)

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

```
;; interp: a runtime environment
;; for cs450js-lang var; same-name
;; vars in front shadow later ones
```

```
;; run: AST -> Result
```

```
(define (run p)
  ;; accumulator env: Environment
  ;; invariant: Contains in-scope variable + result pairs
  (define (run/env p env)
    (match p

           …
                 ))
 (run/env p  ???  ))
```

In-class Coding (prev): **env operations**

- Needed operations:
  - `env-add    : Env Var Result -> Env`
  - `env-lookup : Env Var -> Result`

```
;; An Environment (Env) is one of:
;; - empty
;; - (cons (list Var Result) Env)
;; interp: a runtime environment
;; for cs450js-lang vars; same-name
;; vars in front shadow later ones
```

Think about examples where this happens!

env-add examples

```
;; env-add: Env Var Result -> Env
```

```
(check-equal? (env-add ‘() ‘x 1)
              ‘((x 1)) )        ; empty
```

```
(check-equal? (env-add ‘((x 1)) ‘y 2)
              ‘((y 2) (x 1)) ) ; add new var
```

```
(check-equal? (env-add ‘((x 1)) ‘x 3)
              ‘((x 3) (x 1)) ) ; add shadowed var
```

Env template

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

```
(define (env-fn env … )
  (cond
   [(empty? env) … ]
   [else
    (match-define (cons (list x result) rest-env) env)
     … x … result … (env-fn rest-env … ) … ]))
```

2 cases

2nd case extracts components of compound data

```
;; env-add: Env Var Result -> Env
```

```
(define (env-add env new-x new-res)
  (cond
   [(empty? env) … ]
   [else
    (match-define (cons (list x result) rest-env) env)
     … x … result …(env-add rest-env … ) … ]))
```

```
;; env-add: Env Var Result -> Env
```

```
(define (env-add env new-x new-res)
  (cond
   [(empty? env) (cons (list new-x new-res) env)]
   [else
    (match-define (cons (list x res) rest-env) env)
     … (env-add rest-env … ) … ]))
```

```
;; env-add: Env Var Result -> Env
```

```
(define (env-add env new-x new-res)
  (cond
   [(empty? env) (cons (list new-x new-res) env)]
   [else         (cons (list new-x new-res) env)]))
```

```
;; env-add: Env Var Result -> Env
```

```
(define (env-add env new-x new-res)
  (cons (list new-x new-res) env))
```

env-lookup examples

```
;; env-lookup: Env Var -> Result
```

```
(check-equal? (env-lookup ‘((y 2) (x 1)) ‘x)
              1 ) ; no dup
```

```
(check-equal? (env-lookup ‘((x 2) (x 1)) ‘x)
              2 ) ; duplicate
```

```
(check-equal? (env-lookup ‘() ‘x)
              UNDEFINED-ERROR ) ; not found!
```

```
;; A 450jsResult is one of:
;; - Number
;; - UNDEFINED-ERROR
```

env-lookup

```
;; env-lookup: Env Var -> 450jsResult
```

```
(define (env-lookup env target-x)
  (cond
   [(empty? env) … ]
   [else
    (match-define (cons (list x res) rest-env) env)
     … (env-lookup rest-env … ) … ]))
```

env-lookup: empty (error) case

```
;; env-lookup: Env Var -> 450jsResult
```

```
(define (env-lookup env target-x)
  (cond
   [(empty? env) UNDEFINED-ERROR]
   [else
    (match-define (cons (list x res) rest-env) env)
     … (env-lookup rest-env … ) … ]))
```

env-lookup: non-empty case

```
;; env-lookup: Env Var -> 450jsResult
```

```
(define (env-lookup env target-x)
  (cond
   [(empty? env) UNDEFINED-ERROR]
   [else
    (match-define (cons (list x res) rest-env) env)
     … (env-lookup rest-env … ) … ]))
```

Extract the pieces

env-lookup: non-empty case

```
;; env-lookup: Env Var -> 450jsResult
(define (env-lookup env target-x)
  (cond
   [(empty? env) UNDEFINED-ERROR]
   [else
    (match-define (cons (list x res) rest-env) env)
    (if (var=? x target-x)
        res
     … (env-lookup rest-env … ) … ]))
```

Found target-x

env-lookup: non-empty case

```
;; env-lookup: Env Var -> 450jsResult
```

```
(define (env-lookup env target-x)
  (cond
   [(empty? env) UNDEFINED-ERROR]
   [else
    (match-define (cons (list x res) rest-env) env)
    (if (var=? x target-x)
        res
       (env-lookup rest-env target-x))]))
```

Else, recursive call with remaining env

`run450js` (with an accumulator)

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

```
;; run: AST -> Result
```

```
(define (run p)

  ;; accumulator env: Environment
  (define (run/env p env)
    (match p
      …
     [(var x) (env-lookup env x)]
     [(bind x e body) … (env-add env x (run/env e env)) …]
      …   ))
 (run/env p  ???  ))
```

Environment has **Results** (not **AST**)

How to convert **AST** to **Result**?

Be careful to get correct "**scoping**" (x not visible in expression e, so use unmodified input env)

Bind scoping examples

```
;; A 450jsExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - (list ‘bind Variable Expr Expr)
```

This is called “**lexical**” or “**static**” scoping

Generally accepted to be “best choice” for programming language design
(it’s determined only by program syntax)

We will use this for “CS450js Lang”

```
(check-equal?
  (eval450 ‘(bind x 10 x))
  10 ) ; no shadow
```

Variable reference

```
(check-equal?
  (eval450 ‘(bind x 10 (bind x 20 x)))
  20 ) ; shadow
```

```
(check-equal?
  (eval450
    ‘(bind x 10
       (+ (bind x 20
             x)
          x)) ; 2nd x outof scope here
  30 )
```

Variable references

```
(check-equal?
  (eval450
   ‘(bind x 10
     ‘(bind x (+ x 20)) ; x = 10 here
       x))) ; x = 30 here
   30 )
```

- Repo: **cs450f23/lecture23-inclass**
- File: **bind-examples-**<*your last name*>**.rkt**

In-class Coding 11/27: bind scope examples

Come up with some of your own!

```
(check-equal?
  (eval450 '(bind x 10 x))
  10 ) ; no shadow
```

```
(check-equal?
  (eval450 '(bind x 10 (bind x 20 x))
  20 ) ; shadow
```

```
(check-equal?
  (eval450
   '(bind x 10
      (+ (bind x 20
           x)
         x)) ; 2nd x outof scope here
  30 )
```

```
(check-equal?
  (eval450
   '(bind x 10
     '(bind x (+ x 20)) ; x = 10 here
       x))) ; x = 30 here
   30 )
```

Different Kinds of Scope

(Perl)

- **Lexical (Static)** Scope
  - Variable value determined by **syntactic** code location

```
$a = 0;
sub foo {
  return $a;
}
```

```
sub staticScope {
  my $a = 1; # lexical (static)
  return foo();
}
```

```
print staticScope(); # 0 (from the saved global frame)
```

- **Dynamic** Scope
  - Variable value determined by **runtime** code location
  - Discouraged: **violates "separation of concerns" principal**

```
$b = 0;
sub bar {
  return $b;
}
```

```
sub dynamicScope {
  local $b = 1;
  return bar();
}
```

```
print dynamicScope(); # 1 (from the caller's frame)
```

Other Kinds of Scope

- JS "function scope"
  - `var` declarations
    - follow lexical scope inside functions
    - but <u>not</u> other blocks! (weird?)
  - `let` declarations
    - follow lexical scope inside functions
    - and <u>all</u> other blocks!

```
{
  var x = 2;
}
// x CAN be used here
```

Introduced in ES6 (2015) to fix var weirdness

```
{
  let x = 2;
}
// x can NOT be used here
```

- Global scope
  - Variables in-scope everywhere
  - Added to "initial environment" before program runs

run450js, with an Environment

```
;; run: AST -> Result
(define (run p)

  ;; accumulator env : Environment
  (define (run/e p env)
    (match p
      …
      [(var x) (lookup env x)]
      [(bind x e body) (run/e body (env-add env x (run/e e env)))]
      … ))
 (run/e p  ???  ))
```

3. run body with that new environment

2. add variable x to environment

1. Compute Result that variable x represents

Initial Environment

```
(define (run p)

  ;; accumulator env : Environment
  (define (run/e p env)
    (match p
      …
      [(var x) (lookup env x)]
      [(bind x e body) (run/e body (env-add env x (run/e e env)))]
      … ))
 (run/e p  ???  ))
```

Previously

Initial Environment

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

These don’t need to be separate constructs

Put these into “initial” environment

Initial Environment

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

Put these into “initial” environment

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

```
(define INIT-ENV
`((+ ,450+)
  (- ,450-)))
```

+ variable

Maps to our “450+” function

```
;; A 450jsResult is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function
```

Initial Environment

How do users call these functions???

```
(define INIT-ENV ‘((+ ,450+) (- ,450-)))
```

```
(define (run p)

  ;; accumulator env : Environment
  (define (run/e p env)
    (match p
      …
      [(var x) (lookup env x)]
      [(bind x e body) (run/e body (env-add env x (run/e e env)))]
      … ))
 (run/e p INIT-ENV    ))
```

Function Application in CS450js

```
;; A 450jsExpr (Expr) is one of:
;; - Number
;; - String
;; - Variable
;; - (list ‘bind Var Expr Expr)
;; - (list ‘fncall Expr . List<Expr>)
```

function

arguments

“rest” arg

Specifies arbitrary number of args

(compare with JS “variadic” args)

```
function sum(...theArgs) {
  let total = 0;
  for (const arg of theArgs) {
    total += arg;
  }
  return total;
}
```

Function Application in CS450js: Examples

```
;; A 450jsExpr (Expr) is one of:
;; - Number
;; - String
;; - Variable
;; - (list ‘bind Var Expr Expr)
;; - (list ‘fncall Expr . List<Expr>)
```

function

arguments

```
(fncall + 1 2)
```

Programmers shouldn’t need to write the explicit “fncall”

Function Application in CS450js: Examples

```
;; A 450jsExpr (Expr) is one of:
;; - Number
;; - String
;; - Variable
;; - (list ‘bind Var Expr Expr)
;; - (cons Expr List<Expr>)
```

```
(+ 1 2)
```

Function call case (must be last, why?)

No longer need “rest” arg (why?)

Must be careful when parsing this (HW 8!)

Function Application in CS450js

```
;; A 450jsExpr (Expr) is one of:
;; - Number
;; - String
;; - Variable
;; - (list ‘bind Var Expr Expr)
;; - (cons Expr List<Expr>)
```

parse450js

```
;; A 450jsAST (AST) is one of:
;; - …
;; - (var Symbol)
;; - (bind Symbol AST AST)
;; - (call AST List<AST>)

(struct var [name])
(struct bind [var expr body])
(struct call [fn args])
```

“Running” Function Calls

TEMPLATE: extract pieces of compound data

```
(define (run p)


  (define (run/e p env)
    (match p
       …
     [(call fn args) (apply
                       (run/e fn env)
                       (map (curryr run/e env) args))]

       …
     ))
 (run/e p INIT-ENV))
```

```
;; A 450jsAST (AST) is one of:
;; - …
;; - (var Symbol)
;; - (bind Symbol AST AST)
;; - (call AST List<AST>)

(struct var [name])
(struct bind [var expr body])
(struct call [fn args])
```

“Running” Function Calls

```
(define (run p)


  (define (run/e p env)
    (match p
          ...
      [(call fn args) (apply
                        (run/e fn env)
                        (map (curry??? run/e env) args))]
          ...
      ))
 (run/e p INIT-ENV))
```

TEMPLATE: recursive calls

“Running” Function Calls

How do we actually run the function?

```
;; A 450jsResult is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function
```

```
(define (run p)


  (define (run/e p env)
    (match p
         …
      [(call fn args) (apply ???
                        (run/e fn env)
                        (map (curryr run/e env) args))]
         …
      ))
  (run/e p INIT-ENV))
```

(this only “works” for now)

Function Application in CS450js

```
;; A 450jsExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - (list ‘bind Var Expr Expr)
;; - (cons Expr List<Expr>)
```

Function call case (must be last)

This doesn't let users define their own functions!

Next Feature: **Lambdas?**

“Lambdas” in CS450js

```
;; A 450jsExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - (list ‘bind Var Expr Expr)
;; - (list ‘fn List<Var> Expr)
;; - (cons Expr List<Expr>)
```

CS450js “Lambda” examples

```
;; A 450jsExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - (list ‘bind Var Expr Expr)
;; - (list ‘fn List<Var> Expr)
;; - (cons Expr List<Expr>)
```

```
(fn (x y) (+ x y))
```

```
((fn (x y) (+ x y))
 10 20) ; applied
```

```
(fn (x) (fn (y) (+ x y)) ; “curried”
```

CS450js “Lambda” full examples

```
(check-equal?
  (eval450
   ‘((fn (x y) (+ x y))
     10 20)
   30 )
```

```
(check-equal?
  (eval450
   ‘((bind x 10
     ‘((fn (y) (+ x y))
       20)
   30 ) ; with bind
```

```
(check-equal?
  (eval450
   ‘((bind x 10
     ‘(fn (y) (+ x y)))
     20)
   30 ) ; with bind (fn only)
```

- Repo: **cs450f23/lecture23-inclass**
- File: **fn-examples**-*<your last name>*.rkt

In-class Coding 11/27: fn scope examples

Come up with some of your own!

```
(check-equal?
  (eval450
   ‘((fn (x y) (+ x y))
     10 20)
   30 )
```

```
(check-equal?
  (eval450
   ‘((bind x 10
     ‘((fn (y) (+ x y))
       20)
   30 ) ; with bind
```

```
(check-equal?
  (eval450
   ‘((bind x 10
     ‘(fn (y) (+ x y)))
    20)
   30 ) ; with bind (fn only)
```

CS450js “Lambda” AST node

```
;; A 450jsExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - (list ‘bind Var Expr Expr)
;; - (list ‘fn List<Var> Expr)
;; - (cons Expr List<Expr>)
```

`parse450js`

```
;; A 450jsAST (AST) is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; - (call AST List<AST>)
;; …
(struct fn-ast [params body])
(struct call [fn args])
```

“Running” Functions?

```
;; A 450jsAST (AST) is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; - (call AST List<AST>)
;; …
(struct fn-ast [params body])
(struct call [fn args])
```

```
(define (run p)


  (define (run/e p env)
    (match p
        …
      [(fn-ast params body) ?? params ?? body ??]
        …
      ))
  (run/e p INIT-ENV))
```

What should be the “Result” here?

```
;; A 450jsResult is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function
```

How can we “convert” a 450js program AST into a Racket function???

We can’t!! So we need some other representation

“Running” Functions?

```
;; A 450jsAST (AST) is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; - (call AST List<AST>)
;; …
(struct fn-ast [params body])
(struct call [fn args])
```

How can we “convert” this into a Racket function?

WAIT! Are `fn-val` and `fn-ast` the same?

```
;; A 450jsResult is one of:
;; - …
;; - (Racket) Function
;; - (fn-val List<Symbol> AST ??)
(struct fn-val [params body])
```

We can’t!! So we need some other representation

“Running” Functions? Full example

```
(bind x 10
  (fn (y)
    (+ x y)))
```

`parse450js`

```
(bind ‘x (num 10)
  (fn-ast ‘(y)
    (call (var ‘+)
          (list (var ‘x) (var ‘y)))
```

`run450js`

```
(fn-val ‘(y)
  (call (var ‘+)
        (list (var ‘x) (var ‘y))
```

Now the x is undefined!?

`fn-val` and `fn-ast` <u>cannot</u> be the same!!

“Running” Functions?

```
;; A 450jsAST (AST) is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; - (call AST List<AST>)
;; …
(struct fn-ast [params body])
(struct call [fn args])
```

How can we “convert” this into a Racket function?

WAIT! Are `fn-val` and `fn-ast` the same?

```
;; A 450jsResult is one of:
;; - …
;; - (Racket) Function
;; - (fn-val List<Symbol> AST ??)
(struct fn-val [params body])
```

We can’t!! So we need some other representation

“Running” Functions?

```
;; A 450jsAST (AST) is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; - (call AST List<AST>)
;; …
(struct fn-ast [params body])
(struct call [fn args])
```

A Function `Result` needs an extra environment (for the <u>non-argument variables</u> in the body!)

```
;; A 450jsResult is one of:
;; - …
;; - (Racket) Function
;; - (fn-val List<Symbol> AST Env)
(struct fn-val [params body env])
```

“Running” Functions?

```
;; A 450jsAST (AST) is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; - (call AST List<AST>)
;; …
(struct fn-ast [params body])
(struct call [fn args])
```

```
(define (run p)

  (define (run/e p env)
    (match p
        …
      [(fn-ast params body) ?? params ?? body ??]
        …
     ))
 (run/e p INIT-ENV))
```

What should be the “Result” here?

```
;; A 450jsResult is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function
```

How can we “convert” a 450js program AST into a Racket function???

We can’t!! So we need some other representation

“Running” Functions?

```
;; A 450jsAST (AST) is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; - (call AST List<AST>)
;; …
(struct fn-ast [params body])
(struct call [fn args])
```

```
(define (run p)

  (define (run/e p env)
    (match p
        …
      [(fn-ast params body) ?? params ?? body ??]
        …
      ))
 (run/e p INIT-ENV))
```

```
;; A 450jsResult is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function
;; - (fn-val List<Symbol> AST Env)
(struct fn-val [params body env])
```

“Running” Functions?

```
;; A 450jsAST (AST) is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; - (call AST List<AST>)
;; …
(struct fn-ast [params body])
(struct call [fn args])
```

```
(define (run p)

  (define (run/e p env)
    (match p
      …
     [(fn-ast params body) (fn-val params body env)]
      …
     ))
 (run/e p INIT-ENV))
```

Don’t run body until fn is called

Save the env

```
;; A 450jsResult is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function
;; - (fn-val List<Symbol> AST Env)
(struct fn-val [params body env])
```

Next Time: "Running" Function Calls

How do we actually run the function?

```
;; A 450jsResult is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function
;; - (fn-val List<Symbol> AST Env)
```

```
(define (run p)


  (define (run/e p env)
    (match p

        ...
      [(call fn args) (apply
                        (run/e fn env)
                        (map (curryr run/e env) args))]

        ...
      ))
 (run/e p INIT-ENV))
```

apply doesn't work for **fn-val**!!
must **manually implement "function call"**

(this only "works" for now)

No More Quizzes!

but push your in-class work to:
Repo: **`cs450f23/lecture23-inclass`**