UMass Boston Computer Science

CS450 High Level Languages (section 2)

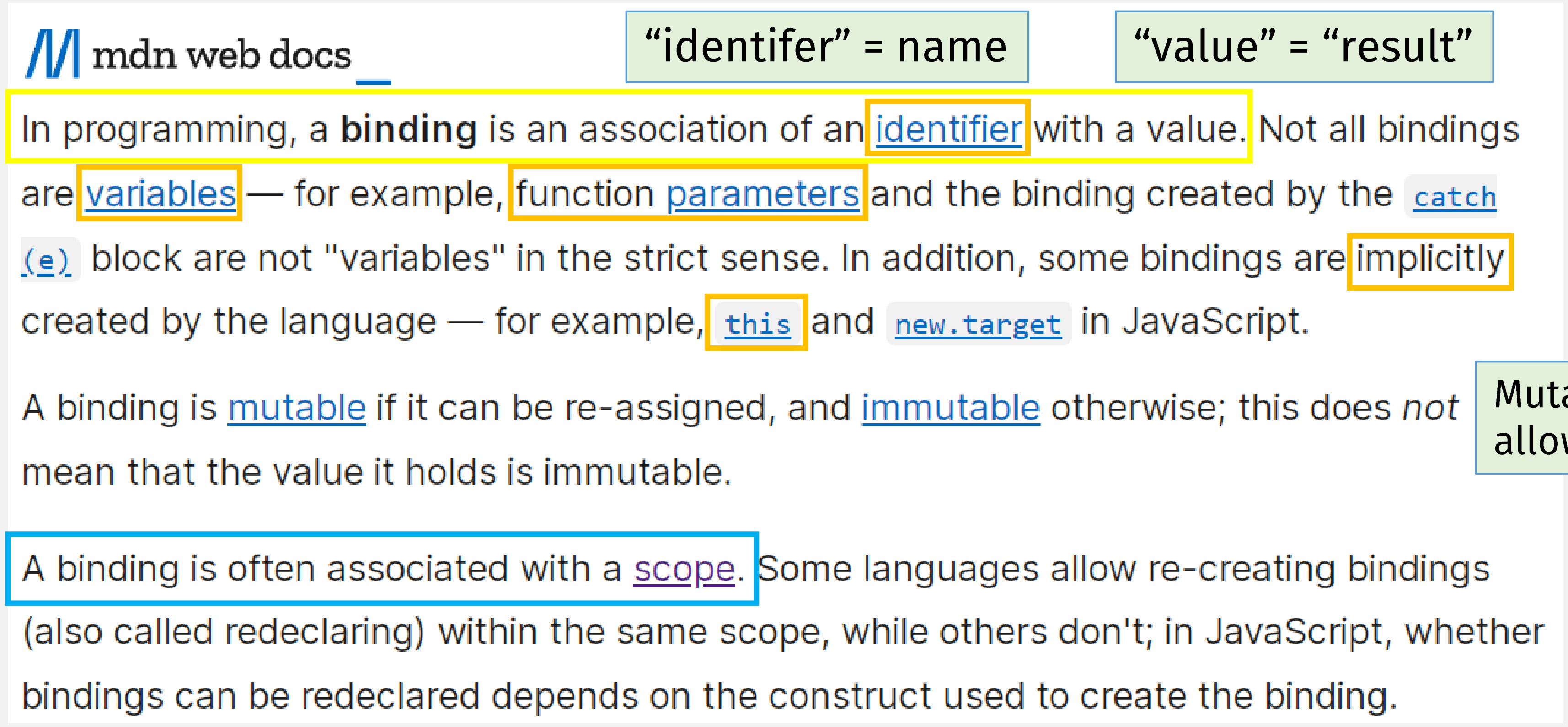
Interpreting Lambda Functions

Wednesday, November 29, 2023

Logistics

- HW 8 out
  - due: **Sun 12/3 11:59 pm EST**
  - **`hw-start`** repo
    - has **`tests-from-lecture23.rkt`** file

Interlude: What is a "binding"?



mdn web docs

"identifer" = name

"value" = "result"

In programming, a **binding** is an association of an identifier with a value. Not all bindings are variables — for example, function parameters and the binding created by the `catch (e)` block are not "variables" in the strict sense. In addition, some bindings are implicitly created by the language — for example, `this` and `new.target` in JavaScript.

A binding is mutable if it can be re-assigned, and immutable otherwise; this does *not* mean that the value it holds is immutable.

A binding is often associated with a scope. Some languages allow re-creating bindings (also called redeclaring) within the same scope, while others don't; in JavaScript, whether bindings can be redeclared depends on the construct used to create the binding.

Mutation (e.g., `set!`) not allowed in this class (so far)

`https://developer.mozilla.org/en-US/docs/Glossary/Binding`

“bind” in “CS450js” Lang: **New Syntax**!

```
;; A Variable (Var) is a Symbol
```

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

Reference a variable binding

Create new variable binding (now with **extra brackets**!)

new binding is in-scope here

CS450jsLang

```
(bind [x 10] (+ x 1))
```

Equivalent to …

Racket

```
(let ([x 10]) (+ x 1))
```

Bind scoping examples

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

bind obeys "**lexical**" or "**static**" scoping

Generally accepted to be "best choice" for programming language design
(bc it's determined only by program syntax)

Var binding

Var reference

```
(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))
  30 )
```

```
(check-equal?
  (eval450
   '(bind [x 10]
     (bind [x (+ x 20)]
      x)))
   30 )
```

Running **bind** programs

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

parse

```
;; A 450jsAST (AST) is one of:
;; …
;; - (bind Symbol AST AST)
;; …
(struct bind [var expr body])
```

run

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

Running **bind**

TEMPLATE : extract pieces

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


  (define (run/e p env)
    (match p
      …
     [(bind x e body)      ??     x    ??    e    ??    body )))]
      … ))
 (run/e p  ???  ))
```

```
;; A 450jsAST (AST) is one of:
;; …
;; - (bind Symbol AST AST)
;; …
(struct bind [var expr body])
```

Running **bind**

TEMPLATE : recursive call

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


  (define (run/e p env)
    (match p
      …
      [(bind x e body) ?? x ?? (run/e e ??) ?? (run/e body ??) ))]
      … ))
 (run/e p  ???  ))
```

```
;; A 450jsAST (AST) is one of:
;; …
;; - (bind Symbol AST AST)
;; …
(struct bind [var expr body])
```

Running **bind**, using environment

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

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

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

Running **bind**, using environment

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

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

1. Compute Result for x (x not in-scope)

Running **bind**, using environment

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

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

2. add x binding to environment

Computes new env
(x in-scope)

Running **bind**, using environment

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

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

3. run body with new env
(x in-scope)

Function Application in CS450js

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

Function call case (must be last, why?)

be careful when parsing this (HW 8!)

What functions can be called?

Function Application in CS450js

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

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

```
;; A 450jsResult (Result) is a:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function
```

What functions can be called?

```
(+ 1 2)
```

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

(Racket) functions, added to initial environment

Function Application in CS450js

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

parse

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

run

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

“Running” Function Calls

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

TEMPLATE: extract pieces of compound data

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

“Running” Function Calls

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

List-processing function

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

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))
```

Runs a Racket function

(this only "works" for now)

Function Application in CS450js

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

What functions can be called?

```
(+ 1 2)
```

```
(??? 1 2)
```

1. (Racket) functions added to initial environment

2. user-defined (“lambda”) functions?

“Lambdas” in CS450js

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

CS450js "Lambda" examples

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

CS450JSLANG

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

Equivalent to …

RACKET

```
(lambda (x y) (+ x y))
```

```
(fn (x) (fn (y) (+ x y)) ; "curried"
```

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

CS450js “Lambda” full examples

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

Expression that evaluates to a function result

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

argument

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

- Repo: **cs450f23/lecture24-inclass**
- File: **fn-examples**-*<your last name>*.rkt

In-class Coding 11/29: fn scope examples

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

Expression that evaluates to a function result

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

argument

Come up with some of your own!

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

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>)
```

parse

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

“Running” Functions?

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

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

TEMPLATE

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

"Running" Functions?

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


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

What should be the "Result" of running a function?

Can we "convert" a 450js program AST into a Racket function???

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

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

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

“Running” Functions?

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

Can we “convert” this into a Racket function?

WAIT! Are **fn-result** and **fn-ast** the same?

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

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

“Running” Functions? Full example

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

parse

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

run

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

Where is the x???

`fn-result` and `fn-ast` cannot be the same!!

(how can we **“remember”** the x)

“Running” Functions?

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

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

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

“Running” Functions?

A Function `Result` needs an extra environment
(for the non-argument variables used in the body!)

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

“Running” Functions?

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


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

    ))
 (run/e p
```

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

What should be the “Result” of running a function?

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

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

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

“Running” Functions?

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


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

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

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

What should be the “Result” of running a function?

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

“Running” Functions?

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

```
(define (run p)



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

body won’t get “run” until the function is called

Remember the current env

"Running" Function Calls: Revisited

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))
```

Runs a Racket function

(this only "works" for now)

"Running" Function Calls: Revisited

How do we actually run the function?

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

```
(define (run p)


  (define (run/e p env)
    (match p
        ...
      [(call fn args) (450apply
                       (run/e fn env)
                       (map (curryr run/e env) args))]
        ...
      ))
  (run/e p INIT-ENV))
```

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

(this doesn't "work" anymore!)

CS450JS Lang "Apply"

Can we refactor data def to make this cleaner?

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : [Racket fn or fn-result] List<Result> -> Result
(define (450apply fn args)
 …
)
```

```
;; A 450jsResult (Result) is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - FnResult
```

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

CS450JS Lang “Apply”

TEMPLATE?

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
  …
)
```

CS450JS Lang “Apply”

TEMPLATE

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?)          …          ] ;; racket function
  [(fn-result params body env)          ;; user-defined function
         …    params            …            body          …            env]))
```

CS450JS Lang “Apply”

TEMPLATE: mutually referential data and template calls!

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?)          …          ] ;; racket function
  [(fn-result params body env)        ;; user-defined function
       …      params    …      (ast-fn body … ) … (env-fn env … ) … ]))
```

env-add

CS450JS Lang “Apply”

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?)        …          ] ;; racket function
  [(fn-result params body env)       ;; user-defined function
      …    (ast-fn body … ) … (env-add env ?? args params ?? ) … ]))
```

These are lists

CS450JS Lang "Apply"

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

(so this function should be inside `run`)

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?)          …          ] ;; racket function
  [(fn-result params body env)          ;; user-defined function
      …     (ast-fn body … ) … (foldl env-add env params args) … ]))
```

`run/e`

CS450JS Lang “Apply”

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?)               ???
                                 ...         ] ;; racket function
  [(fn-result params body env)      ;; user-defined function
   (run/e body (foldl env-add env params args))]))
```

CS450JS Lang “Apply”

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?) (apply fn args)] ;; racket function
  [(fn-result params body env)     ;; user-defined function
   (run/e body (foldl env-add env params args))]))
```

Runs a Racket function

WAIT! What if the the number of params and args don’t match!

CS450JS Lang "Apply"

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?) (apply fn args)] ;; racket function
  [(fn-result params body env)     ;; user-defined function
   (if (= (length params) (length args))
       (run/e body (foldl env-add env params args))
       …   ]))
```

CS450JS Lang "Apply": arity error

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?) (apply fn args)] ;; racket function
  [(fn-result params body env)     ;; user-defined function
   (if (= (length params) (length args))
       (run/e body (foldl env-add env params args))
       ARITY-ERROR)]))
```

```
;; A 450jsResult (Result) is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - ARITY-ERROR
;; - FnResult
```

No More Quizzes!

but push your in-class work to:
Repo: **`cs450f23/lecture24-inclass`**