Representation Choices

$3 + 3 = 6$

[num C (n : number)]

- 32bit int } predefined
- float
- list of bits } unbounded
- list of characters

```
[closV (arg : symbol)
       (body : ExprC)
       (env : Env)]
```

- create closures
- apply them
  - environments
    or
  - substitution

```
[closV' (f : Value → Value)]
```

in interp:

```
[lamC (a b)
      (closV' (lambda (arg) (interp b (extend-env (bind a arg)
                                                  env))))]
```

```
[appC (f a) ((closV-f (interp f env))
                      (interp a env))]
```

Syntactic interp:
interprets based the structure of the syntax

meta interpreter:
interprets using constructs in host language

meta-circular: everything "cheats"

```
(define-type Value
  [numV (n : number)]
  [closV (arg : symbol) (body : ExprC) (env : Env)])

(define (interp [expr: ExprC] [env : Env]) : Value
  (type-case ExprC expr
    [numC (n) (numV n)]
    [idC (n) (lookup n env)]
    [plusC (l r) (num+ (interp l env) (interp r env))]
    [multC (l r) (num* (interp l env) (interp r env))]
    [appC (f a) (local ([define f-value (interp f env)])
                  (interp (closV-body f-value)
                          (extend-env (bind (closV-arg f-value)
                                            (interp a env))
                                      (closV-env f-value))))]
    [lamC (a b) (closV a b env)]))
```

```
(define-type Value
  [numV (n : number)]
  [closV (f : (Value -> Value))])

(define (interp [expr: ExprC] [env : Env]) : Value
  (type-case ExprC expr
    [numC (n) (numV n)]
    [idC (n) (lookup n env)]
    [plusC (l r) (num+ (interp l env) (interp r env))]
    [multC (l r) (num* (interp l env) (interp r env))]
    [appC (f a) (local ([define f-value (interp f env)]
                        [define a-value (interp a env)])
                  ((closV-f f-value) a-value))]
    [lamC (a b) (closV (lambda (arg-val)
                         (interp b
                           (extend-env (bind a arg-val) env))))]))
```

Right → left eval order

→ (local ([define r' (interp r env)]
[define l' (interp l env)])
(num+ l' r'))

Choices for environments

- list of bindings
  - binding : symbol x Value

```
(define (lookup x env)
  (type-case Env env
    [mtEnv ... Some error here...]
    [extend (y v env')
            (if (symbol=? y x)
                v
                (lookup x env'))]
```

Env : symbol → Value

```
(if (lookup x env) (env x))

(define mtEnv' (λ (y) error))
(define extend' (λ (y v env')
  (λ (x) (if (symbol=? y x) v
             (env' x)))))
```