```
(define (valof expr env)
  (match expr
    [n #:when (number? n) n]
    [y #:when (symbol? y)
       (unbox (apply-env env y))]
    [`(λ(,x),b)
     (λ(a) (valof b (extend-env x a env)))]
    [`(set! ,y ,e) (set-box! (apply-env env y)
                             (valof e env))]
    [`(,rator ,y) #:when (symbol? y)
     ((valof rator env) (box (unbox (apply-env env y))))]
    [`(,rator ,rand)
     ((valof rator env) (box (valof rand env)))]))
```

★ 1
Sol. CDV.

```
(define (valof expr env)                                   CBR
  (match expr
    [`,n #:when (number? n) n]
    [`,y #:when (symbol? y)
         (unbox (apply-env env y))]
    [`(λ (,x) ,b)
     (λ (a) (valof b (extend-env x a env)))]
    [`(set! ,y ,e) (set-box! (apply-env env y)
                             (valof e env))]
    [`(,rator ,y) #:when (symbol? y)
     ((valof rator env) (apply-env env y))]
    [`(,rator ,rand)
     ((valof rator env) (box (valof rand env)))]
```

"CBReference" yes

Can mutating parameters in a subroutine affect the calling procedure?

"CBV" no

```
;; create a local
; var  <- parameter to subr.
((λ(x)
  ((λ(y) (begin2 (set! y 7))     } Subroutine
     x))
 x)
5)
```

X -> [5]
CBV: 5   y -> [~~5~~ 7]   } CBV behavior

CBR: 7   X -> [~~5~~ 7] <- y   } CBR behavior

CBV — alias in parameters to subroutines?

CBR

Not strict

?

Cbname | cbv

Strict
do evaluate the arg b4 passing to closure

do we evaluate the argument before applying the function?

```
(define (valof expr env)   ; CBName
  (match expr
    [n #:when (number? n) n]
    [y #:when (symbol? y)
       ((unbox (apply-env env y)))]
    [`(λ(,x),b)
     (λ(a)(valof b (extend-env x a env)))]
```

```
[`(rator ,y) #:when (symbol? y)
  ((valof rator env)                    (apply-env env y))]
[`(,rator ,rand)
   ((valof rator env)(box(λ()(valof rand env))))]))
```

> ((λ(x) 120)

(Ω)

in OBN 120 !

$\Omega = ((\lambda(x)\,(x\,x))\,(\lambda(x)\,(x\,x)))$

inf loop.

> ((λ(x) 120)

(! lots))

((λ(x) (+ x x x x)

(! big #))

```
(Δ (valof expr env)                    ~~CBname~~ CBneed
  (match expr
    [n #:when (number? n) n]
    [y #:when (symbol? y)
       (unbox-need (apply-env env y))]
    [`(λ (,x) ,b)
     `(λ (a) (valof b (extend-env x a env)))]
```

```
[`(rator ,y) #:when (symbol? y)
  ((valof rator env)                    (apply-env env y))]
[`(,rator ,rand)
   ((valof rator env) (box (λ () (valof rand env))))]))
```

```
(△ (unbox-need b)
   (let ((val ((unbox b))))
     (begin
       (set-box! b (λ() val))
       val)))
```

memoization

(random , nexp) (random (valof nexp end))

>((λ(x) (+ x x))
(random 121))

in CBneed, value of this is even