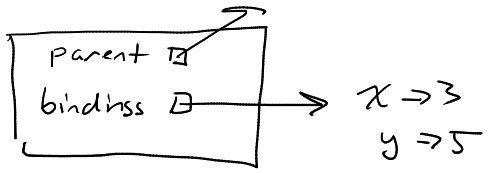
Frames - purpose: local variables

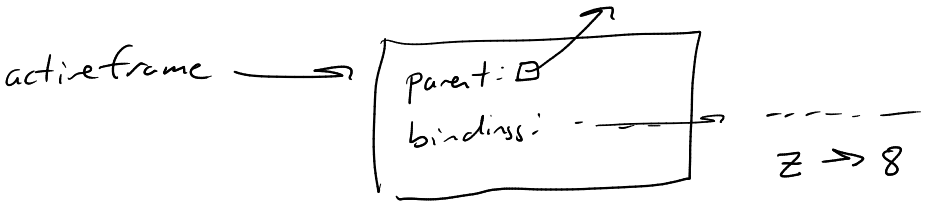



```
(let ((x 3) (y 5))
  x)
```

What does let do?

- → creates a new frame
- → parent points to active frame when let was called
- → assigns local variables

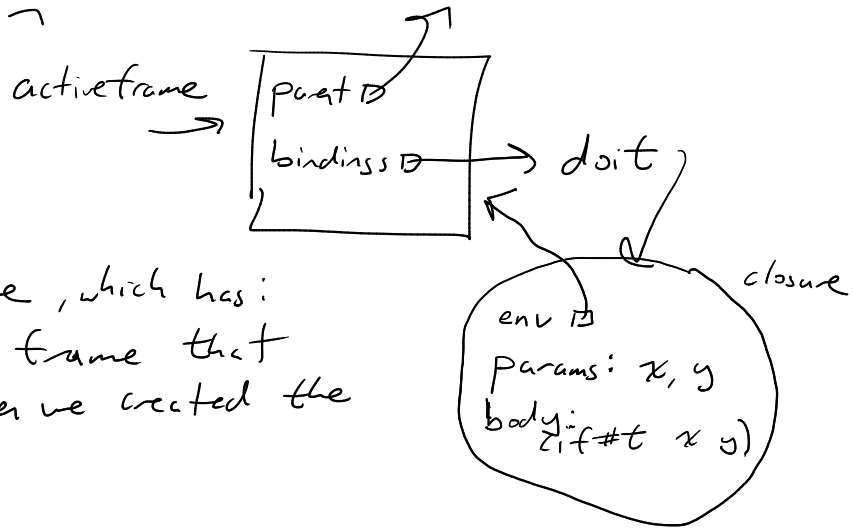
(define z 8) - adds a binding to the active frame



---

lambda creates a function

```
(define doit
  (lambda (x y)
    (if #t x y)))
```



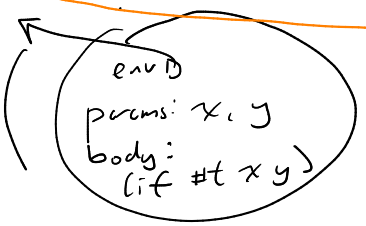
→ creates a closure, which has:

1. a ptr to the frame that was active when we created the closure
2. parameters
3. code (body)

How to evaluate a function

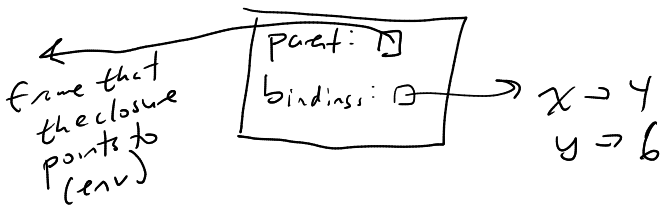
```
(doit (+ 3 1) (+ 4 2))
```

- evaluates each item seperately



4 6

→ evaluate the function:

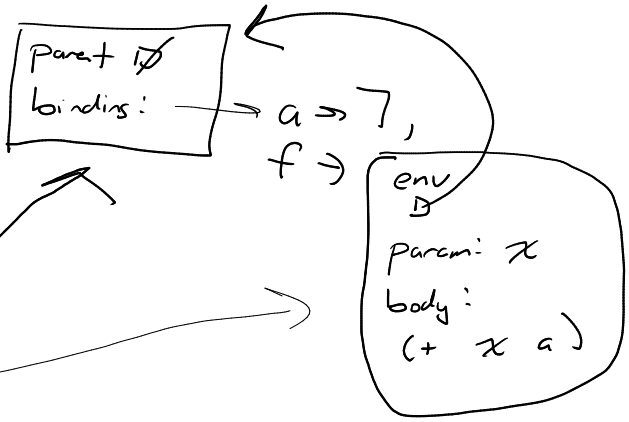


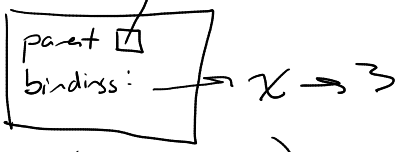
1. Create a new frame
   - bindings are the parameters
   - parent - env ptr in the closure
2. evaluate code in the body

```
(define a 7)
(define f
  (lambda (x)
    (+ x a)))
(f 3)
```

↳ 10

global frame





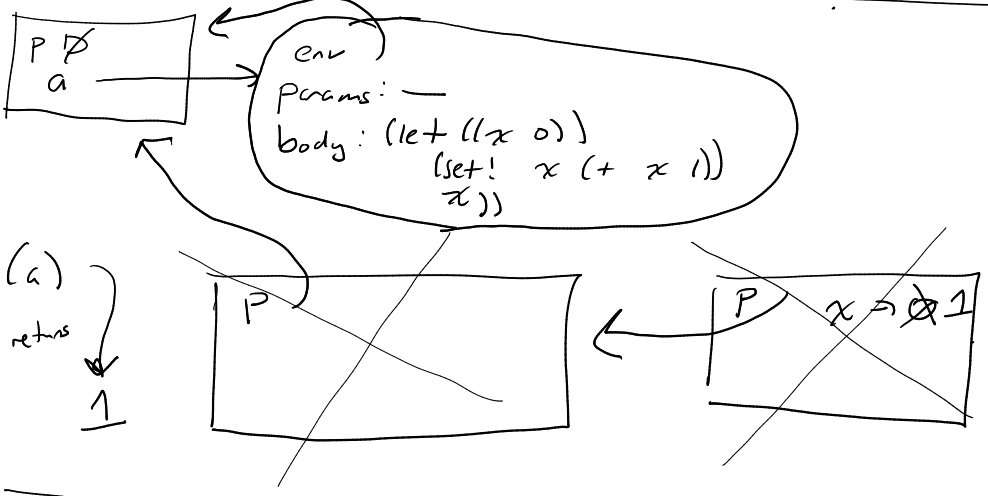
(+ x a)
3 7 → 10

```
(define a
  (lambda ()
    (let ((x 0))
      (set! x (+ x 1))
      x)))
```

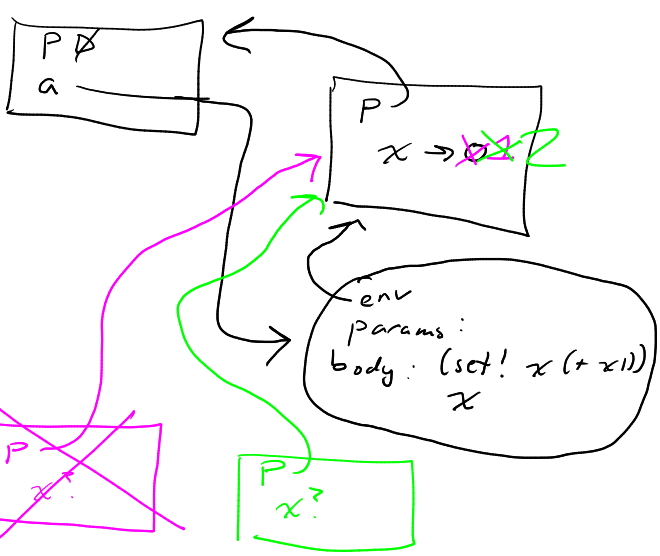
(set! z 5)

recassign z to be 5

```
(define a
  (let ((x 0))
    (lambda ()
      (set! x (+ x 1))
      x)))
```



```
(define a
  (let ((x 0))
    (lambda ()
      (set! x (+ x 1))
      x)))
```



(a) → 1

(a) → 2