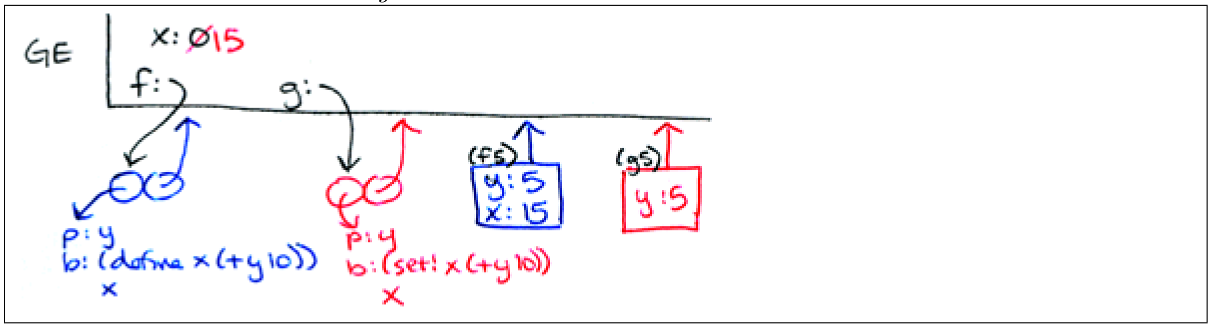
Scoping, `define` versus `set!`, and Shadowing

```
(define x 0)
(define f
  (lambda (y)
    (define x (+ y 10))
    x))
(define g
  (lambda (y)
    (set! x (+ y 10))
    x))
```

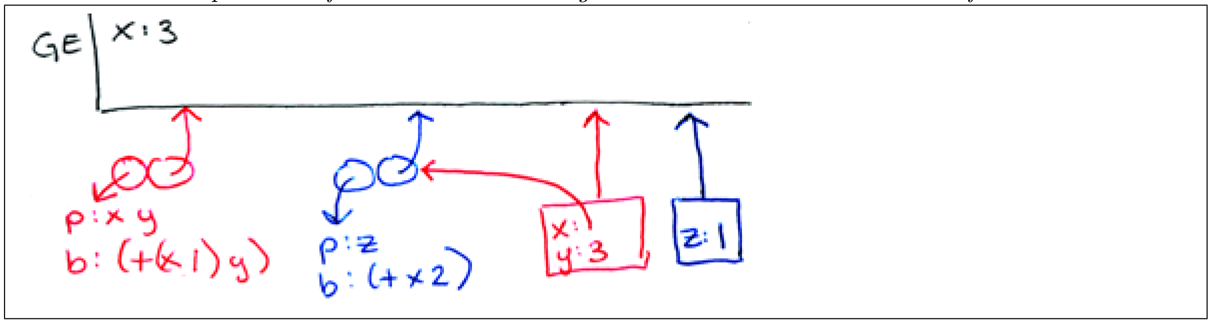
Find the values of:
`(f 5)` $\boxed{15}$, `x` $\boxed{0}$, `(g 5)` $\boxed{15}$, and `x` $\boxed{5}$
...and show the environment diagram:



Nameless Wonders

```
(define x 3)
((lambda (x y) (+ (x 1) y))
 (lambda (z) (+ x 2))
 3)
```

What is the value of this expression? $\boxed{8}$
Show all relevant portions of the environment diagram used to evaluate this block of code.



Aspartame (Desugaring `let`)

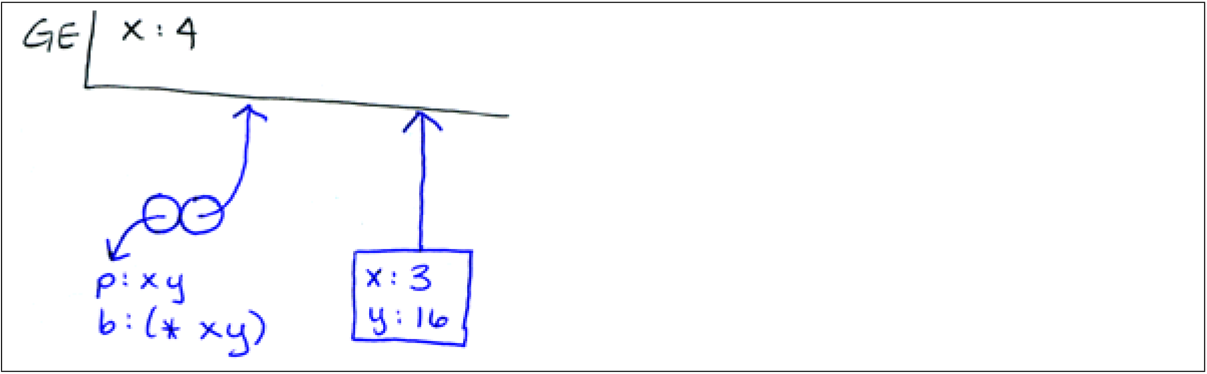
Desugar the following expression:

```
(define x 4)
(let ((x (+ 2 1))
      (y (square x)))
  (* x y))

;DESUGARS TO:

((lambda (x y) (* x y))
 (+ 2 1)
 (square x)) ;==> 48
```

Show all relevant portions of the environment diagram used to evaluate this block of code.

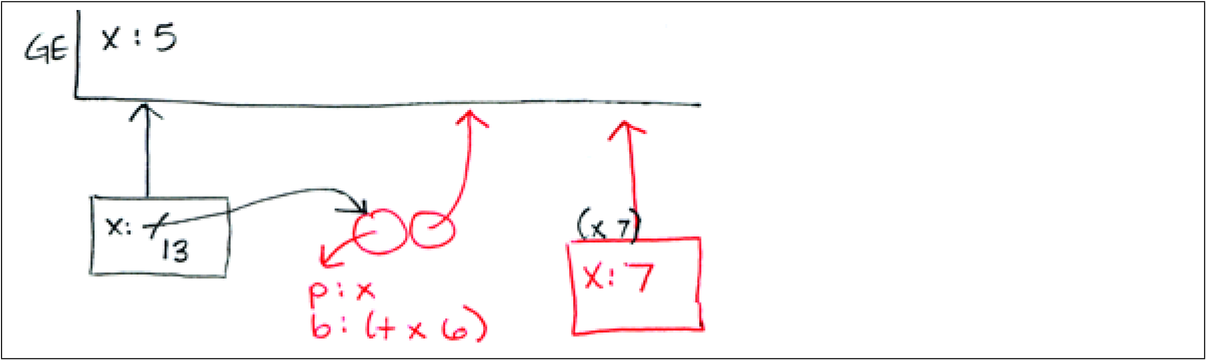


λ-`let`

```
(define x 5)
(let ((x (lambda (x) (+ 6 x))))
  (set! x (x 7))
  x)
```

What is the value of this expression? $\boxed{13}$

Show all relevant portions of the environment diagram used to evaluate this block of code.

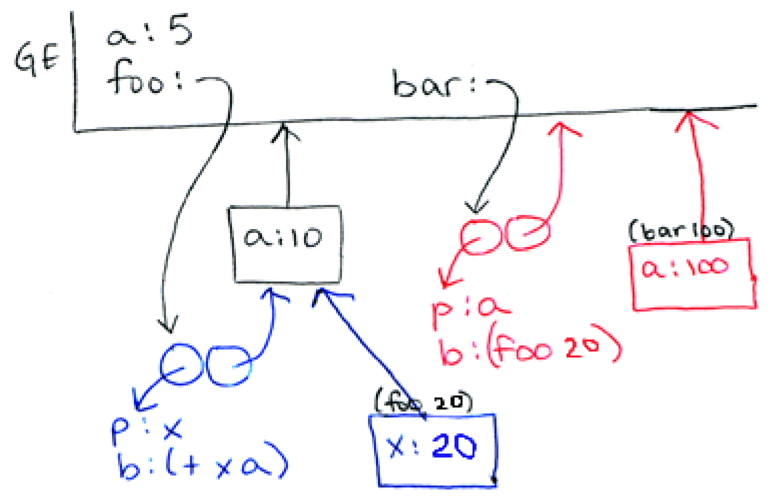


Yet More Complexity

```
(define a 5)
(define foo
  (let ((a 10))
    (lambda (x)
      (+ x a))))
(define (bar a) (foo 20))
(bar 100)
```

What is the value of this expression? $\boxed{30}$

Show all relevant portions of the environment diagram used to evaluate this block of code.



Insanity!

```
(define (make-count-proc f)
  (let ((count 0))
    (lambda (x)
      (if (eq? x 'count)
          count
          (begin (set! count (+ count 1))
                 (f x))))))

(define sqrt* (make-count-proc sqrt))
(define square* (make-count-proc square))
```

Find the values of:

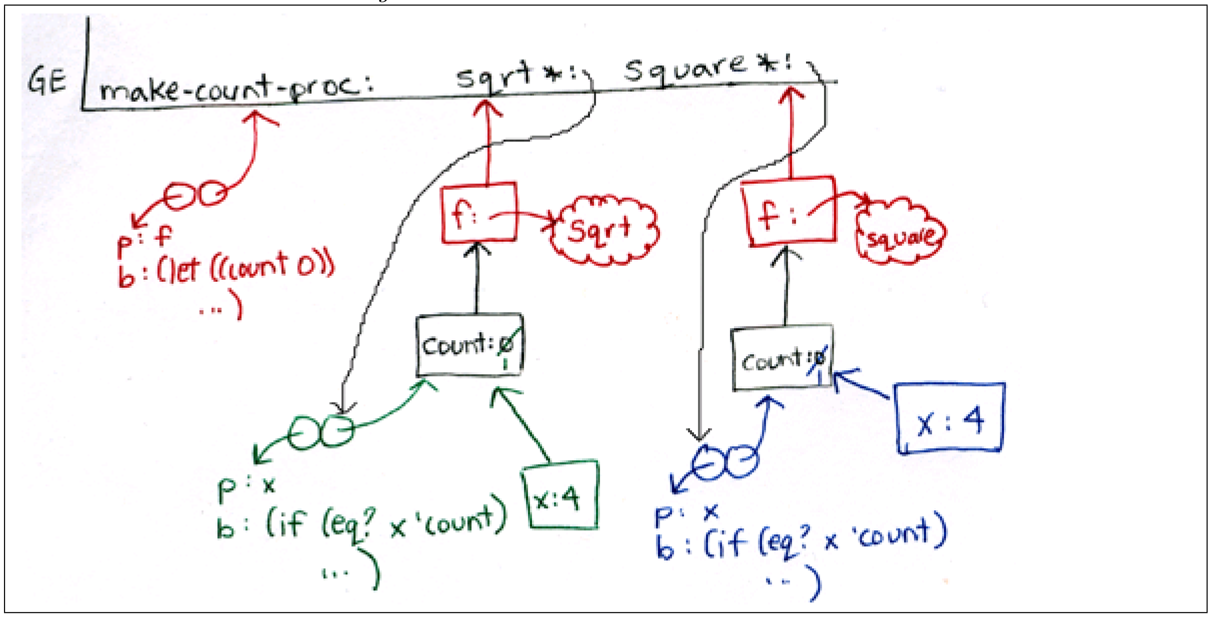
`(sqrt* 4)` $\boxed{2}$,

`(sqrt* 'count)` $\boxed{1}$,

`(square* 4)` $\boxed{16}$,

and `(square* 'count)` $\boxed{1}$

...and show the environment diagram:


GE
make-count-proc:
sqrt*:
square*:
p: f
b: (let ((count 0))
...)
f:
sqrt
f:
square
count: ~~0~~ 1
count: ~~0~~ 1
x:4
x: 4
p: x
b: (if (eq? x 'count)
...)
p: x
b: (if (eq? x 'count)
...)