Lambda Calculus - 100

What the following code returns

```
(let ((a 3))
  (let ((a 4)
        (b a))
    (list a b)))
```

What is `(4 3)`

Lambda Calculus - 200

The value returned by this expression:

```
((lambda (x f)
   (f (f x)))
 3
 (lambda (y)
   (+ y y)))
```

What is `12`

Lambda Calculus - 300

Given the following definition of f:

```
(define f
  (lambda (x)
    (x (lambda (y) (* y 2)))))
```

It's the expression which, when f is applied to it, returns 6.

What is: `(lambda (z) (z 3))`

OR: `(lambda (z) 6)`

Lambda Calculus - 400

A function, that when applied to itself, returns a function, that when applied to 17 returns 17.

What is: `(lambda (x) x)`

OR: `(lambda (f) (lambda (y) 17))`

Lists - 100

The printed representation in Scheme of the following box and pointer diagram:

What is `((b c d) a c d)`

Lists - 200

The expression returned by the following code:

```
(define x '(a b x))
(define y (list x x (list 'x x)))
(set-cdr! (cdr y) (list 'w))
y
```

What is `((a b x) (a b x) w)`

Lists - DAILY DOUBLE

If we were to implement cons, car, and cdr as procedures, by writing cons as a procedure of its two arguments, like so:

```
(define (cons x y)
  (lambda (m) (m x y)))
```

then this is how "cdr" would be defined.

What is

```
(define (cdr l)
  (l (lambda (x y) y)))
```

Lists - 400

The missing expressions in this following definition

```
(define (accumulate f init lst)
  (if (null? lst)
      init
      (____________ ____________
         (accumulate f init
           (cdr lst)))))
```

What are `f` and `(car lst)`