CSSE 304 SRT Review Session 9/29/2024 6-8pm

Answer the following by completing the procedures.

A *sum-of-evens:*
Takes a list of numbers, returns the sum of the even elements in the list.

```
(define sum-of-evens
  (lambda (a)
     (apply + (filter even? a))))
```

Example: `(sum-of-evens '(12 3 2 8 5)) → 22`

B *sum-of-max*:
Takes a list of lists of numbers, and returns the sum of the max element in each sublist.

```
(define sum-of-max
  (lambda (a)
     (apply + (map (lambda (x) (apply max x)) a))))
```

Example: `(sum-of-max '((12 13 16) (1 3) (2))) → 21`

C *to-first-items*:
Takes a single procedure as an argument, and returns a procedure that, when given a list of lists, returns that list with the passed-in procedure applied to the first item in each sublist.

```
(define sum-of-evens
  (lambda (a)
     (lambda (b)
       (map (lambda (x) (cons (a (car x)) (cdr x))) b))))
```

Example: `((to-first-items add1) '((1 2) (2 'banana) (5)))`
`→ ((2 2) (3 'banana) (6))`

Let Lambda Fun!

The following Code uses let and lambda in some really interesting ways, write the output of the following code.

```
(define foo
  (let ((x 10))
    (lambda ()
      ~~(let ((y 0 ))~~
        (lambda (z)
          (set! x (- x z))
          (set! y (+ y z))
          (display (list x y)))))))

(define one (foo))
(define two (foo))

(one 2)
(two 5)
(one 2)
```

one: x=[illegible]
y=4
two: x=[illegible]
y=5

'(8, 2)
'(3, 5)
'(1, 4)

Determine the Bound and Unbound Variables in the Following Statement

Consider the lambda calculus expression:

b b

`(y (lambda (x) (x (z (lambda (y) (lambda (x) (y))))))))`

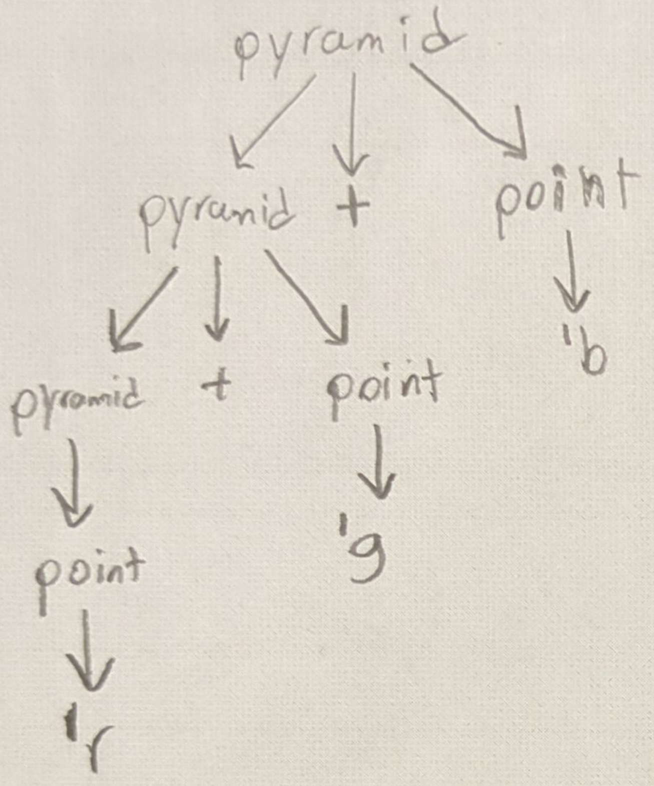
Which variables appear bound X, Y

Which variables appear free Y, Z

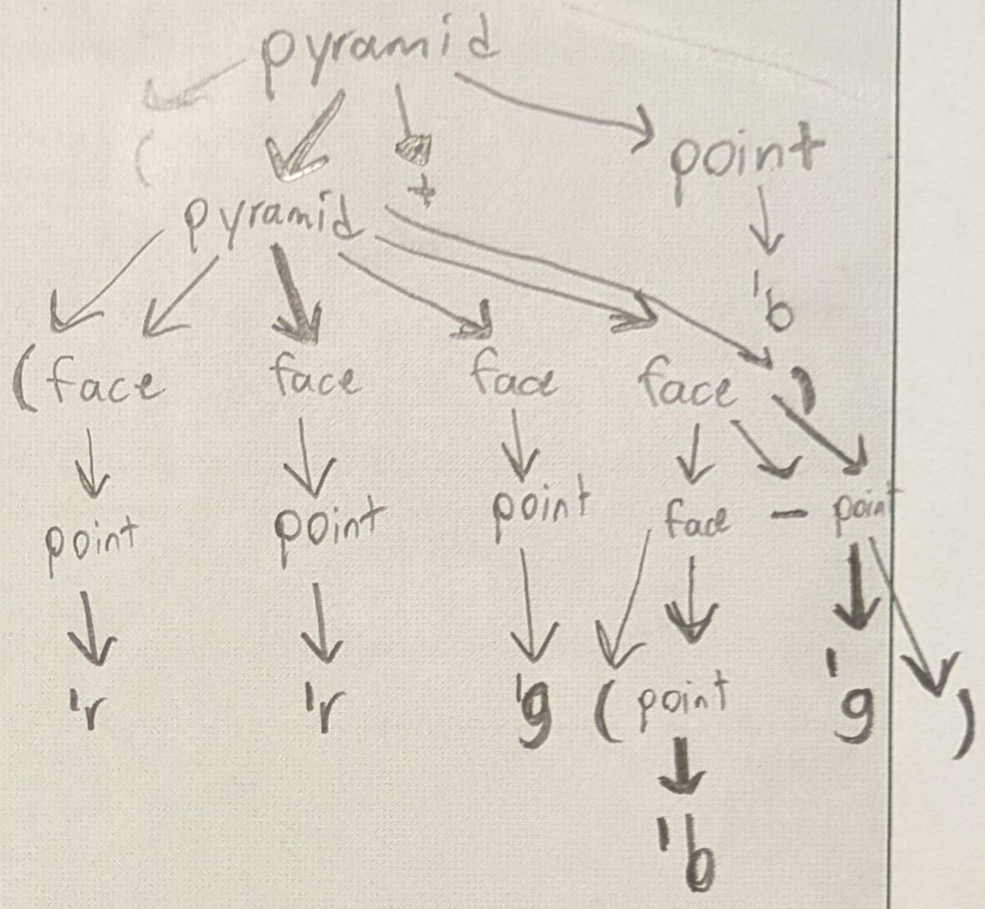
Complete Derivation Trees for the Following Syntax:

```
<pyramid> ::= <pyramid> + <point> | (<face> <face> <face> <face>) | <point>
<face> ::= (<face> - <point>) | <point>
<point> ::= 'r | 'g | 'b
```

```
'r + 'g + 'b
```



```
('r 'r 'g ('b - 'g)) + 'b
```


pyramid
+
point
pyramid
'b
(face
face
face
face
)
point
point
point
face
-
point
'r
'r
'g
(
point
'g
)
'b