1.
Let: parallel blinding
Forms a set of identifier that can only be used in "the let body"

```
> (let ([me "Bob"])
    me)

"Bob"
```

let* : Sequential Binding
is like let except that you are able to use their id for later expressions. It is like having nested lets in the same expression

```
(let* ([x (list "Burroughs")]
       [y (cons "Rice" x)]
       [z (cons "Edgar" y)])
    (list x y z))
```

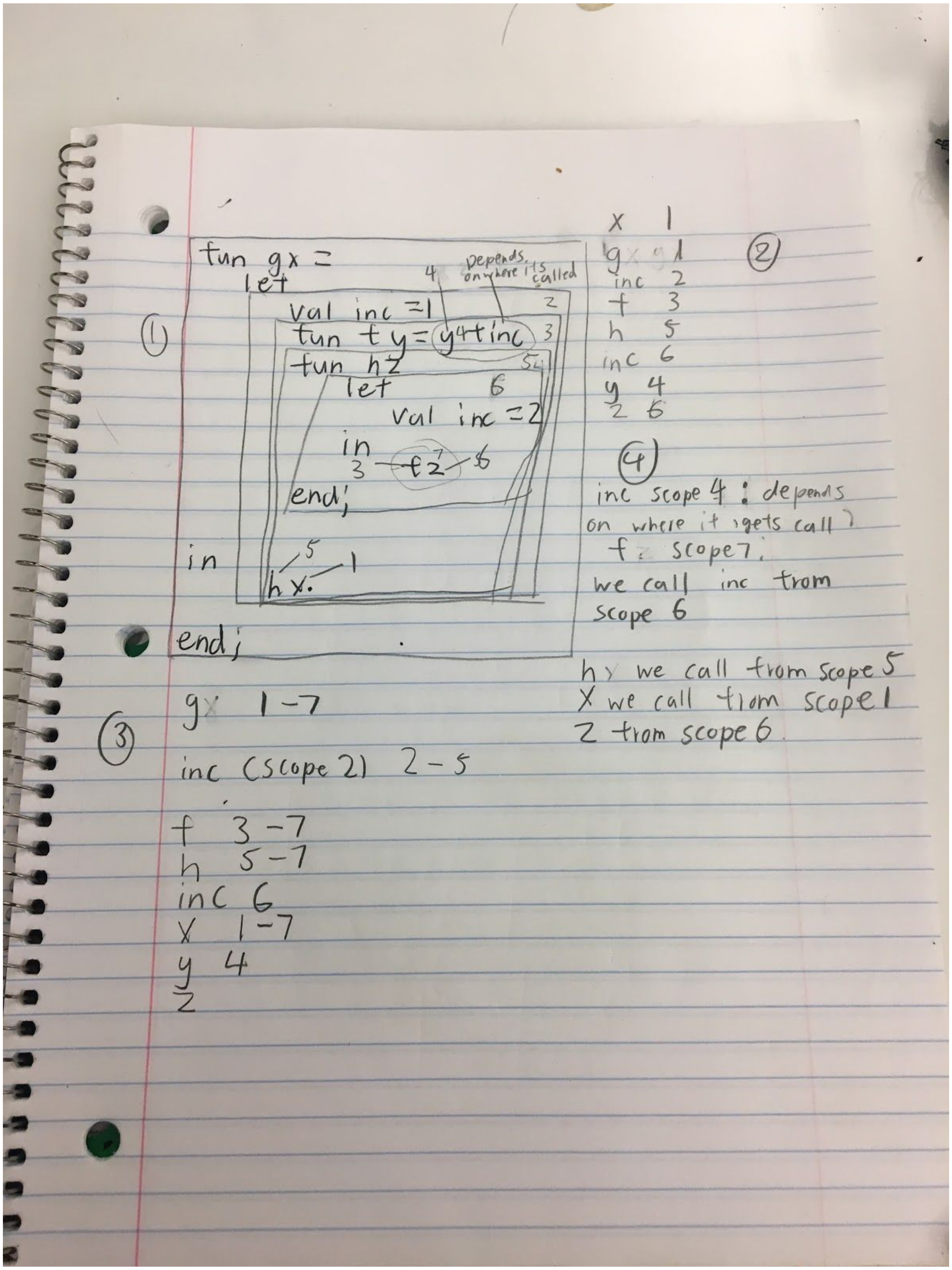
Letrec: Recursive Binding
These blocks can be used on the entire scope even earlier expressions

```
(letrec ([swing
          (lambda (t)
            (if (eq? (car t) 'tarzan)
                (cons 'vine
                      (cons 'tarzan (cddr t)))
                (cons (car t)
                      (swing (cdr t))))))])
    (swing '(vine tarzan vine vine)))

'(vine vine tarzan vine)
```

Source: https://docs.racket-lang.org/guide/let.html

2.


fun g x =
let
val inc = 1
fun f y = y + inc
fun h z
let
val inc = 2
in
f z
end;
in
h x
end;
depends on where it's called
①
x 1
g 1
inc 2
f 3
h 5
inc 6
y 4
z 6
②
③
g 1-7
inc (scope 2) 2-5
f 3-7
h 5-7
inc 6
x 1-7
y 4
z
④
inc scope 4 : depends on where it gets call?
f : scope 7.
we call inc from scope 6
h we call from scope 5
x we call from scope 1
z from scope 6

3

1. No since the function f does not use any thing that is not within its local state
2. Yes since the variable that F needs is from its local state
3. Unknown since we do not know what variables F needs ie what variables it might need to reference.

4 Ruby supports pass in functions
The way you do it is by
```
`def format_all_the_things(&formatter)
  one = formatter.call("thing 1")
  two = formatter.call("thing 2")
  [one, two].join " & "
end`
```
Source:(http://blog.jessitron.com/2013/03/passing-functions-in-ruby-harder-than.html)

This language has known error relating to the address that the function gets return to and you would get local jump error since the function gets its own stack.