Rule 1:

Every function **definition** (including anonymous function definitions) creates a new closure where

- the code part of the closure points to the function's code
- the environment part of the closure points to the frame that was current when the function was defined (the frame we are currently using to look up variables)

Rule 2:

Every function **call** creates a new frame where

- the new frame's table contains bindings for all of the function's arguments and their corresponding values
- the new frame's pointer points to the same frame that the function closure's environment pointer points to.
    - That is, if the definition of a function f created a closure where the environment part of the closure points to a frame R, then whenever f is called, a new frame is created that will point to R as well.

Rule 2a:

Every **evaluation of a let expression** creates a new frame where

- the new frame's table contains bindings for all of the let expression's variables and their corresponding values
- the new frame's pointer points to the frame where the let expression was defined

---

Practice:

```
(define (f g) (let ((x 3)) (g 2)))
(define x 4)
(define (h y) (+ x y))
(define z (f h))
```

PROGRAM 1

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

(define g (f 3)

(define z (g 4)
```

PROGRAM 2

```
(define (f g)
  (let ((x 2))
    (g x)))

(define (h y)
  (let ((x 3))
    (lambda (y) (+ x y))))

(define m (f h))
(define n (m 7))
```