Lab 2 Notes: Higher order Functions

① Higher order Functions & currying

- Functions can return functions!

```
def outer():
    def inner():
        return "something"
    return inner
```

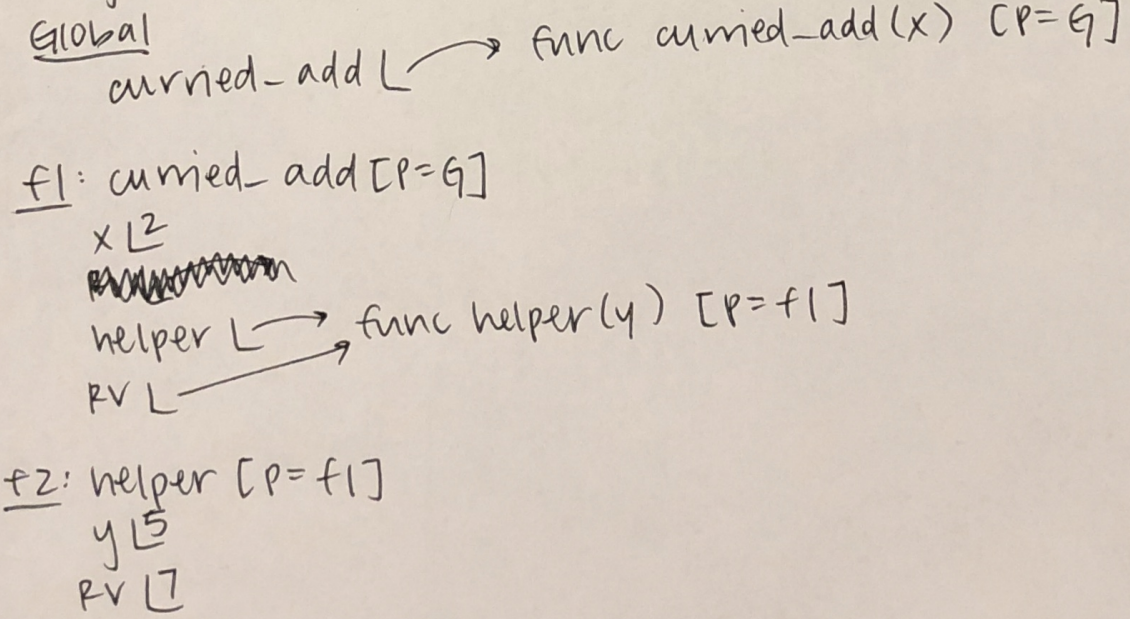
```
>>> outer()
Function
>>> outer()()
      inner
"something"
```

- currying: use higher order functions to convert function that takes multiple args into a chain of functions that each take a single arg

```
def curried_add(x):
    def helper(y):
        return x+y
    return helper
```

```
>>> curried_add(2)(5)
      f1
7     f2
```

- Env diagram:



② Lambda Expressions

- Lambda expr evaluate to functions. They specify:
  1) parameters
  2) return expr

- Syntax: `lambda <parameters> : <return expr>`

  EX: `lambda x : x`
  A function that takes x and returns x

- Properties of lambda ~~functions~~ expr:
  - An expression that evaluates to a value. Evaluating the lambda expr does NOT create/modify variables
  - Creates an anonymous lambda function with no intrinsic name
  - Note that we can assign a variable to be equal to a lambda expression

```
f = lambda x, y : (x + y) * 2
```

```
>>> f(3, 5)
16
```

```
>>> (lambda x, y : (x + y))(3, 5)
16
```

Env. Diagram Example:

```
1  def compose(f, g):                    r.v. f4
2                                          r.v. f3
3      return lambda x: f(g(x))
4  f = compose(lambda x: x*x,            r.v. f1
5              lambda y: y+1)
6  result = f(12)
```

r.v. f2

Global

compose L ⟶ func compose(f, g) [P = G]

f L (f1)

result L 169 (f2)

f1: compose [P = G]

f L ⟶ func λ(x) <line 4> [P = G]

g L ⟶ func λ(y) <line 5> [P = G]

RV L ⟶ func λ(x) <line 2> [P = f1]

f2: λ <line 2> [P = f1]

x L 12

RV L 169

f(r.v. f3) → f(13) → r.v. f4

f3: λ <line 5> [P = G]

y L 12

RV L 13

f4: λ <line 4> [P = G]

x L 13

RV L 169