

# Discussion 02: More Environments and Recursion

.....

TA: **Jerry Chen**

Email: **[jerry.c@berkeley.edu](mailto:jerry.c@berkeley.edu)**

TA Website: **[jerryjrchen.com/cs61a](http://jerryjrchen.com/cs61a)**

# Agenda

1. Attendance
2. Feedback!
3. Announcements
4. Check Your Understanding
5. Lambdas (skip, view slides later)
6. (Quick) Higher order functions, again
7. More environment diagrams
8. Recursion

## **Thanks for your feedback! Some common trends:**

Too much talking, not enough "doing"

- I will blab a bit less

Pace is too fast/too slow

- Might have to compromise a bit here

Hungry

- For knowledge!
- (and food, probably)
- Me too

# Attendance

Sign in at [bit.do/jerrydisc](https://bit.ly/jerrydisc)

OR

Please put your name, SID, and email on the sign-in sheet.

# Announcements

**Hog due Thursday (Today!)**

**Homework 3 due Tuesday**

**Midterm 1 Thursday, 9/15**, look for a Piazza post

- Alternate exam requests close on Sunday
- Discussion next week?

**Guerrilla Section this Saturday**

**Sign ups for CSM sections are open!** Sections start the week after the midterm

# Check Your Understanding

I made a mistake last week with explaining boolean order:

```
>>> True and 1 or not 1/0 and False
```

# Check Your Understanding

```
square = lambda x: x * x
```

```
def test(f, x):  
    if f(x) % 2 == 0:  
        return lambda g, x: g(square, x)  
    else:  
        return f(x)
```

```
print(test(lambda s: s // 2, 20)(test, 7))
```

# Environment Diagrams

**New:** Values can also be bound to functions!

Some rules:

- **Function call: create and number new frame** (f1, f2, etc.)  
— always start in global frame
- **Assignment:** write variable name and expression value
- **Def statements:** record function name and bind function object. Remember parent frame!
- **Frames return values** upon completion (Global is special)



# Environment Diagrams

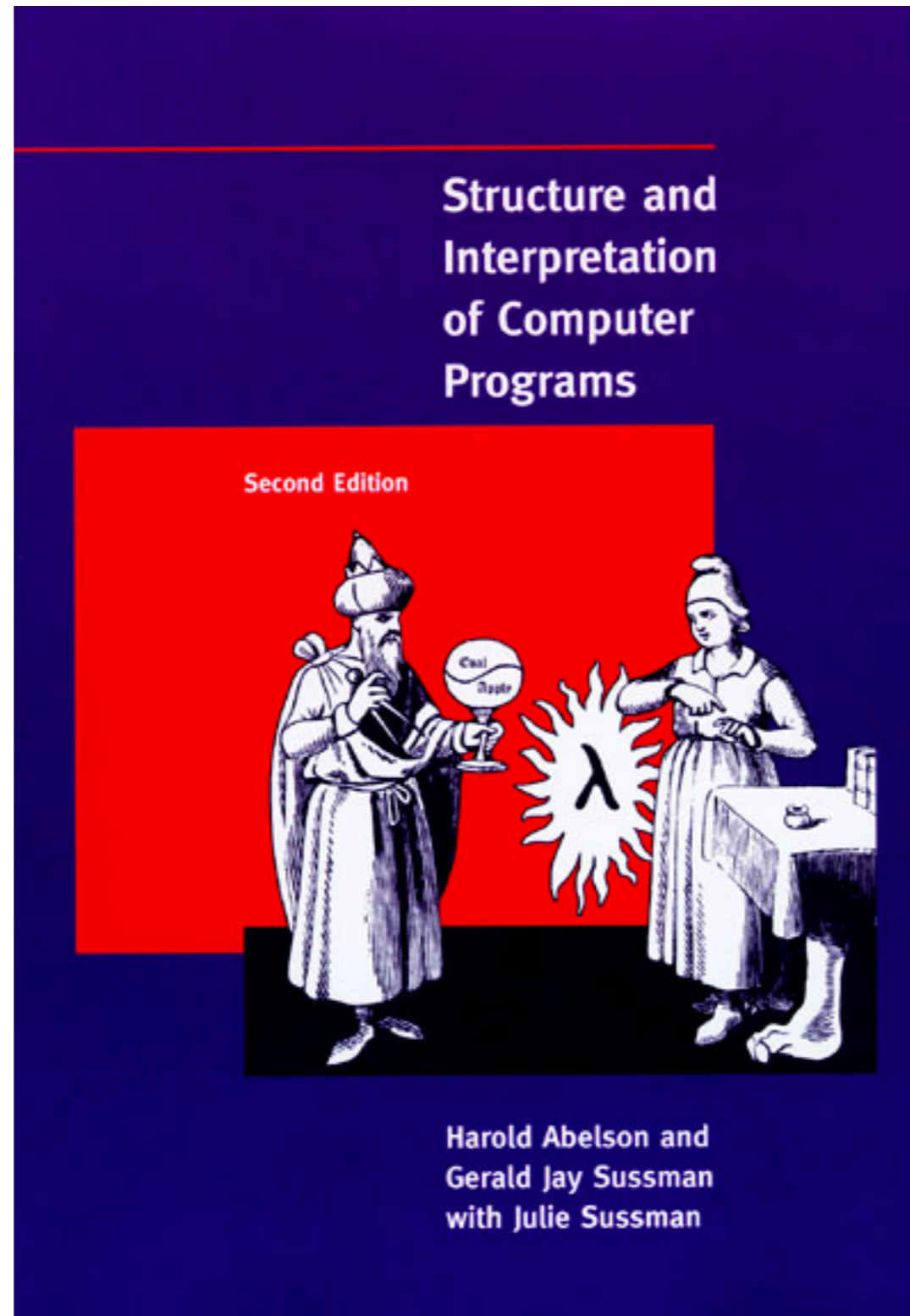
From Kevin Chen's Fall 2015 Review (<https://goo.gl/Z6GNwi>)

```
x = 2
def dread(pirate):
    x = 30
    def roberts(westley):
        x = 400
        return westley + pirate(x)

    return roberts(x)

dread(lambda spot: x + spot)
```

# A Lambda Detour



# A Lambda Detour

```
(lambda x, y: x + y * y)(4, 5)
```

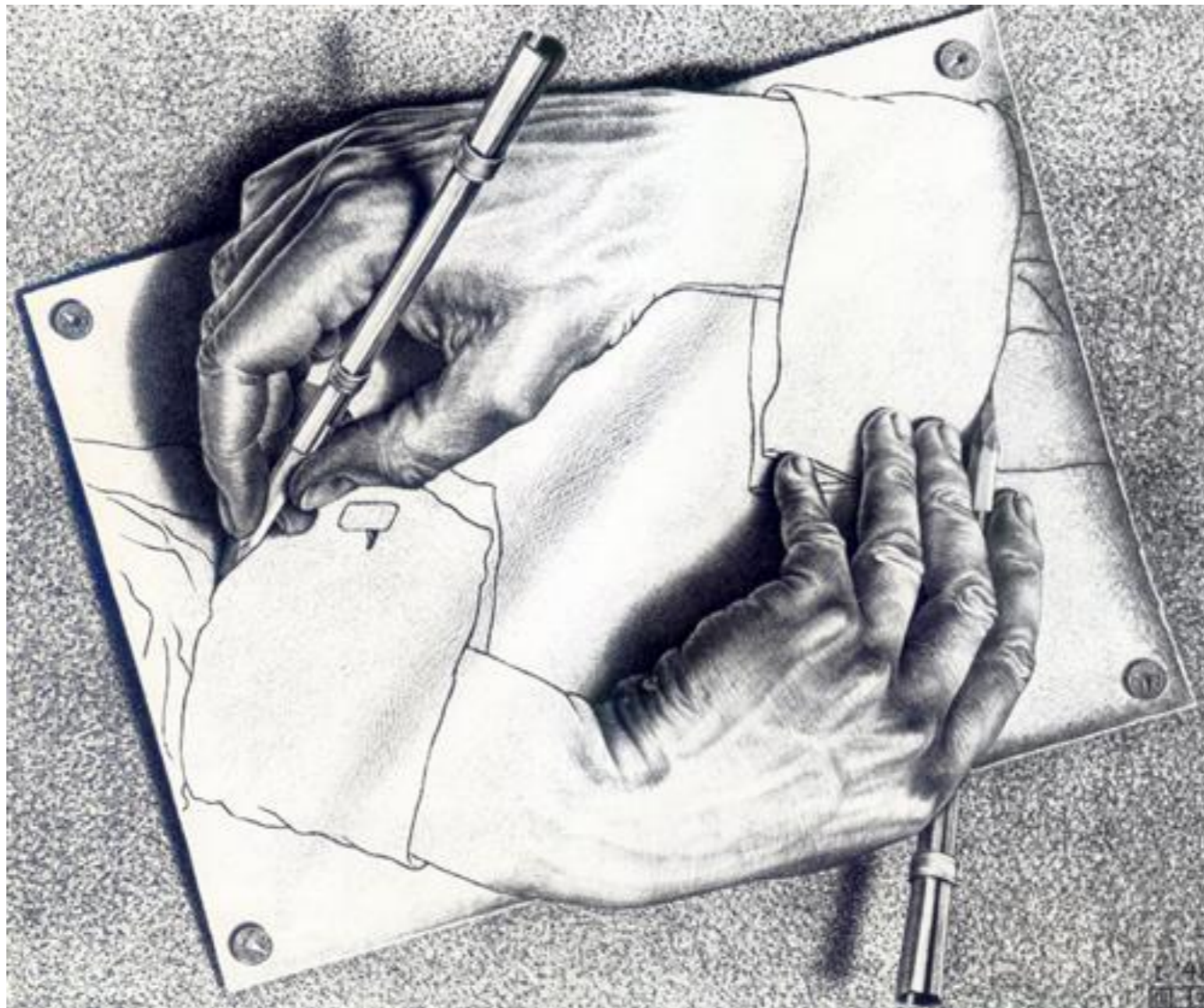
Lambda definition

Lambda call

Result (after currying):

```
(lambda x = 4, y = 5: x + y * y)
```

# Recursion



*Drawing Hands* by M. C. Escher

# Recursion

Components of a recursive function

- **Base case**, a simple stopping condition
- **Recursive calls** on smaller problem
- **Putting it together**: solve our prob using recursive result

**Leap of faith**: assume our recursive function solves any simpler version of the problem

# Recursion

Factorial example

```
def factorial(n):  
    if n == 0:  
        return 1  
    return n * factorial(n - 1)
```

# Recursion

Fast Exponentiation

```
def exp(b, n):  
    if n == 0:  
        return 1  
    if n % 2 == 0:  
        return exp(b ** 2, n / 2)  
    else:  
        return b * exp(b, n - 1)
```

# Recursion

What's Wrong?

```
def hailstone (n) :  
    print (n)  
    if n == 1:  
        return  
    elif n % 2 == 0:  
        n = n // 2  
        hailstone (n - 1)  
    else:  
        n = 3 * n + 1  
        hailstone (n - 1)
```



# Tree Recursion

Recursive functions can sometimes require more than one call!

$$\text{Fib}(n) = \text{Fib}(n - 1) + \text{Fib}(n - 2)$$

Very powerful, but also potentially very slow (why?)

Useful when you want to represent choices (e.g. taking one stair or two stairs)