# 61A Lecture 5

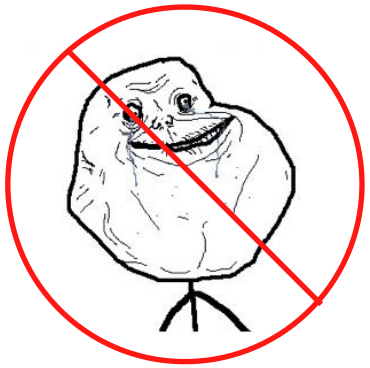
Wednesday, September 10

# Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - http://cs61a.org/hw/released/quiz1.html
  - 3 points; graded for correctness
  - Submit in the same way that you submit homework assignments
  - If you receive 0/3, you will need to talk to the course staff or be dropped
  - Open computer & course materials, but no external resources such as classmates
  - Practice quiz from Fall 2013: http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html
- "Practical Programming Skills" DeCal starts Thursday 9/11, 6:30pm to 8pm in 306 Soda
  - http://42.cs61a.org, run by Sumukh Sridhara (TA)
- Guerrilla Section 1 on Higher-order functions: Saturday 9/13, 12:30pm to 3pm in 306 Soda
- Homework 2 (which is small) due Monday 9/15 at 11:59pm.
- Project 1 (which is BIG) due Wednesday 9/17 at 11:59pm.

# Office Hours: You Should Go!

**You are not alone!**



http://cs61a.org/staff.html

# Environments for Higher-Order Functions

# Environments Enable Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value

**Higher-order functions:**

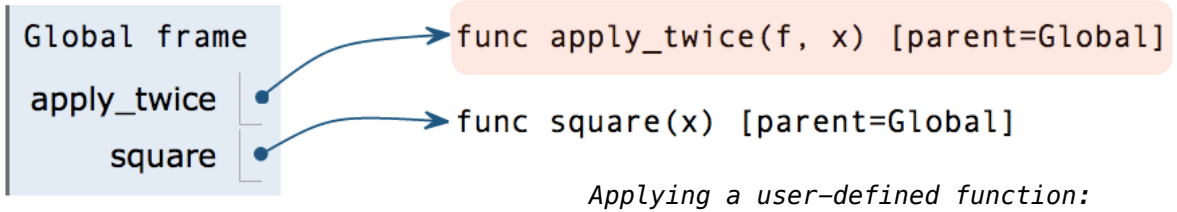
- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions

*Environment diagrams describe how higher-order functions work!*

(Demo)

# Names can be Bound to Functional Arguments

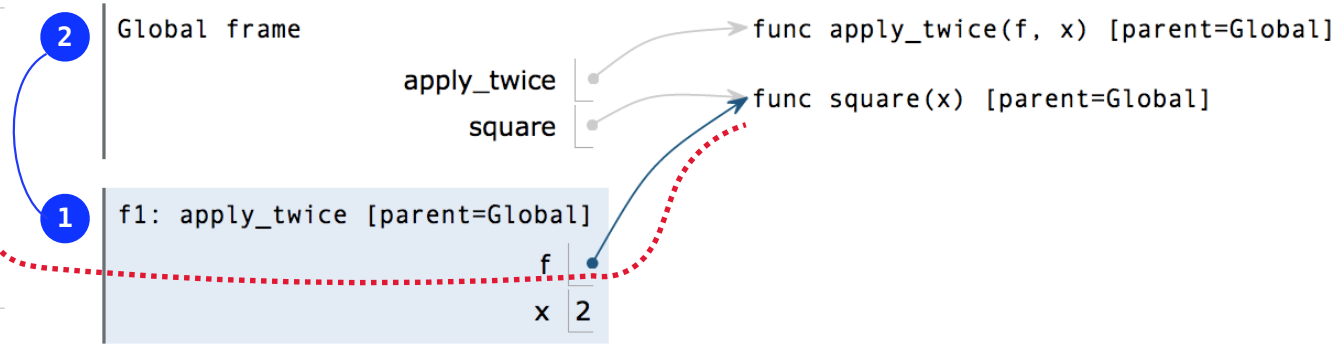
```
1  def apply_twice(f, x):
2      return f(f(x))
3
4  def square(x):
5      return x * x
6
7  result = apply_twice(square, 2)
```




*Applying a user-defined function:*

- Create a new frame
- Bind formal parameters (f & x) to arguments
- Execute the body: return f(f(x))

```
1  def apply_twice(f, x):
2      return f(f(x))
3
4  def square(x):
5      return x * x
6
7  result = apply_twice(square, 2)
```




Interactive Diagram

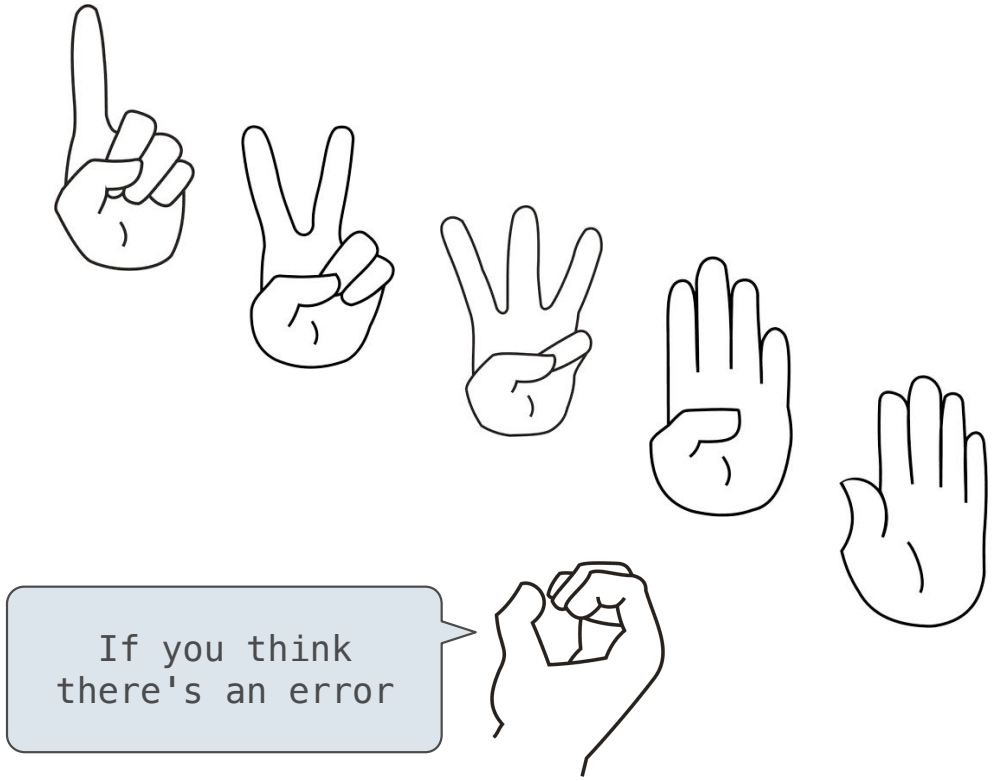
# Discussion Question

What is the value of the final expression below? (Demo)

```
def repeat(f, x):
    while f(x) != x:
        x = f(x)
    return x

def g(y):
    return (y + 5) // 3

result = repeat(g, 5)
```




Interactive Diagram

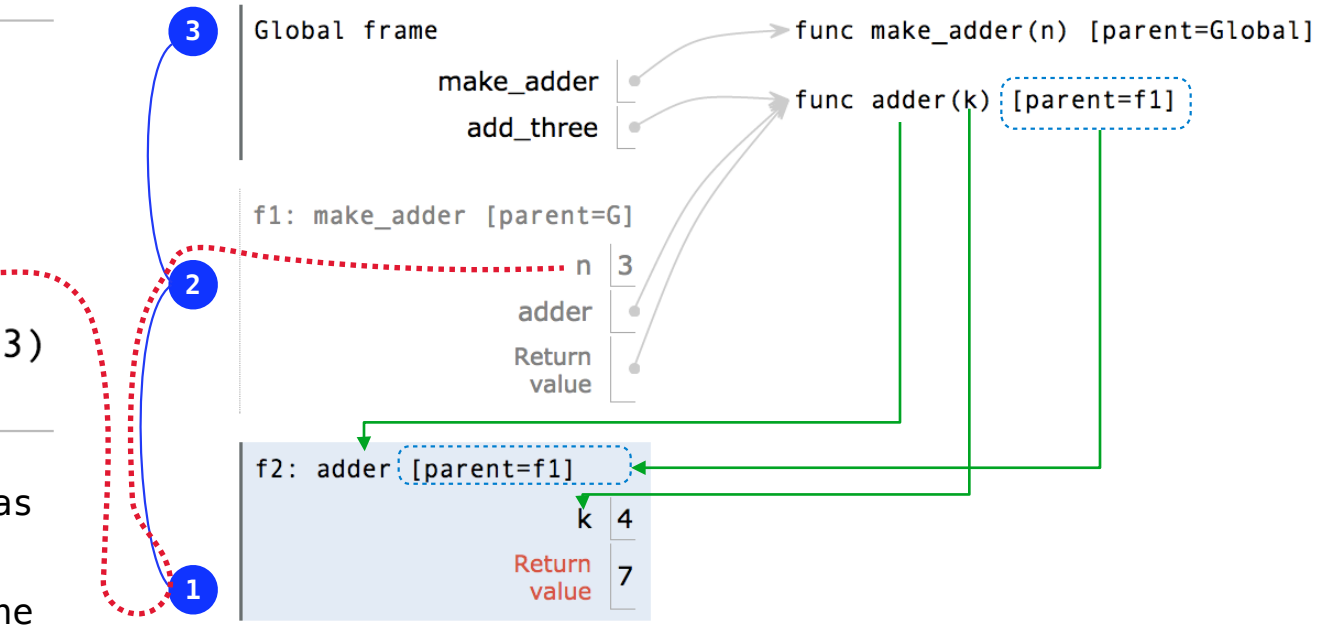
# Environments for Nested Definitions

(Demo)

# Environment Diagrams for Nested Def Statements

Nested def

```
1 def make_adder(n):
2     def adder(k):
3         return k + n
4     return adder
5
6 add_three = make_adder(3)
7 add_three(4)
```




- Every user-defined function has a parent frame (often global)
- The parent of a function is the frame in which it was defined
- Every local frame has a parent frame (often global)
- The parent of a frame is the parent of the function called

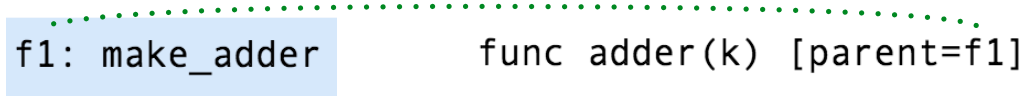
Interactive Diagram

# How to Draw an Environment Diagram

**When a function is defined:**

Create a function value: `func <name>(<formal parameters>) [parent=<label>]`

Its parent is the current frame.




Bind `<name>` to the function value in the current frame

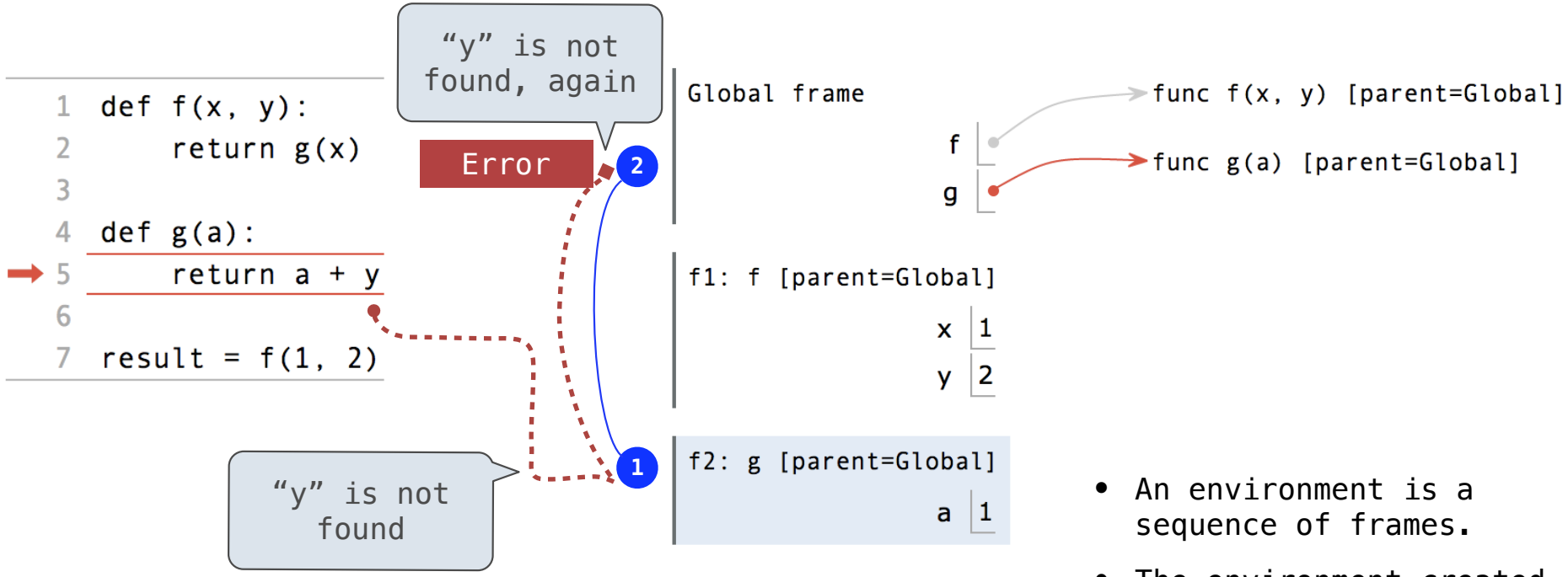
**When a function is called:**

1. Add a local frame, titled with the `<name>` of the function being called.
2. Copy the parent of the function to the local frame: `[parent=<label>]`
3. Bind the `<formal parameters>` to the arguments in the local frame.
4. Execute the body of the function in the environment that starts with the local frame.

# Local Names

(Demo)

# Local Names are not Visible to Other (Non-Nested) Functions




- An environment is a sequence of frames.
- The environment created by calling a top-level function (no def within def) consists of one local frame, followed by the global frame.

Interactive Diagram

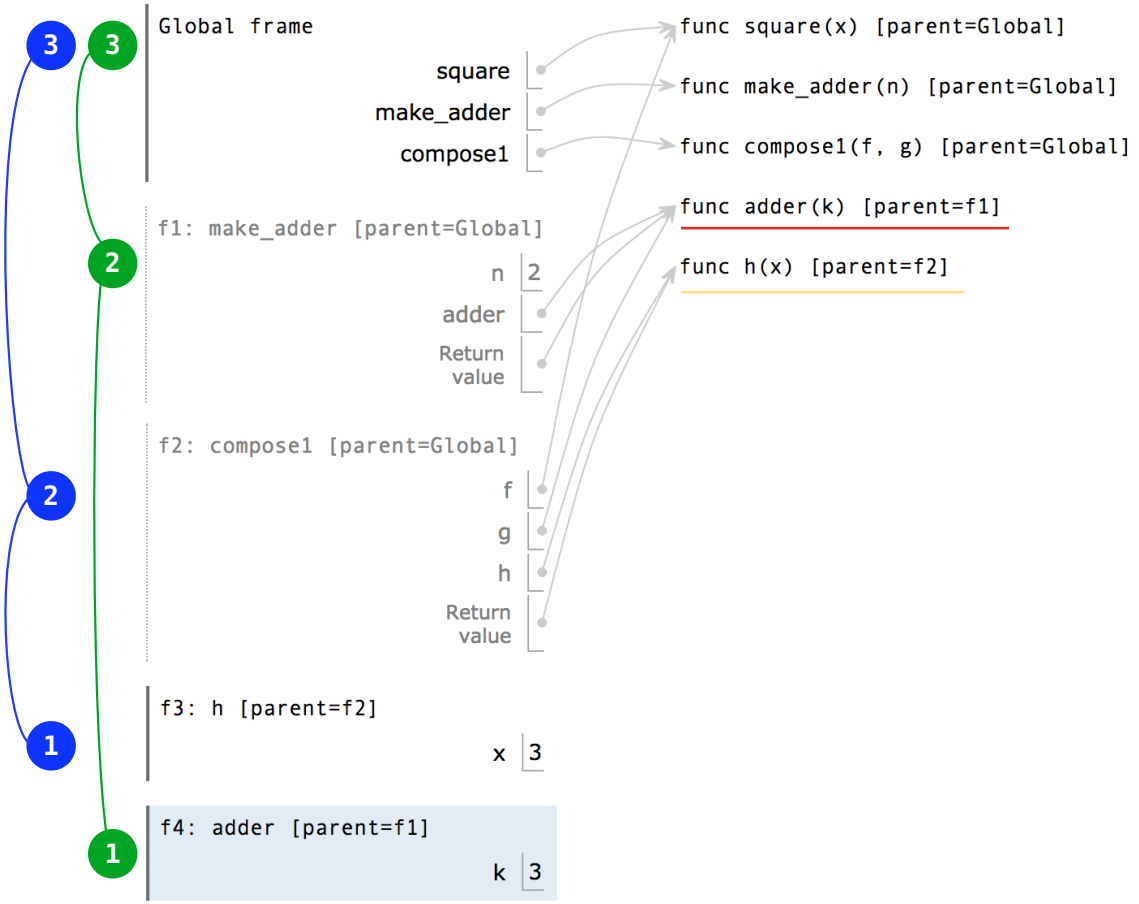
# Function Composition

(Demo)

# The Environment Diagram for Function Composition

```
1  def square(x):
2      return x * x
3
4  def make_adder(n):
5      def adder(k):
6          return k + n
7      return adder
8
9  def compose1(f, g):
10     def h(x):
11         return f(g(x))
12     return h
13
14 compose1(square, make_adder(2))(3)
```

Return value of make_adder is an argument to compose1




Interactive Diagram