

## 61A Lecture 5

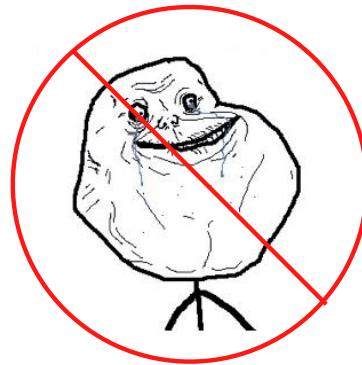
---

## Announcements

## Office Hours: You Should Go!

---

**You are not alone!**



<http://cs61a.org/office-hours.html>

## Environments for Higher-Order Functions

## Environments Enable Higher-Order Functions

---

**Functions are first-class:** Functions are values in our programming language

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

*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)
```

Global frame  
apply\_twice  
square

func apply\_twice(f, x) [parent=Global]

func square(x) [parent=Global]

*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)
```

2 Global frame

1 f1: apply\_twice [parent=Global]

apply\_twice  
square

func apply\_twice(f, x) [parent=Global]

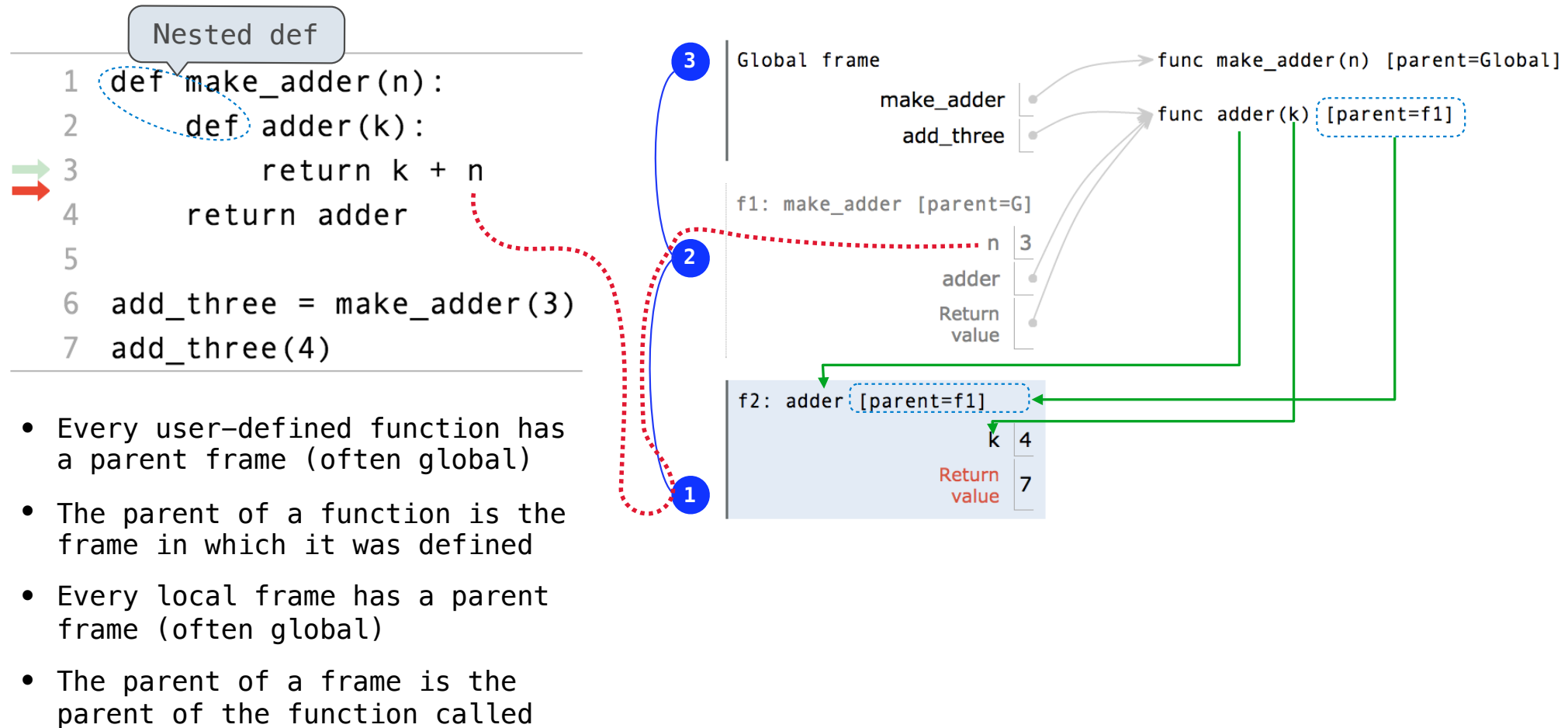
func square(x) [parent=Global]

f  
x 2

# Environments for Nested Definitions

(Demo)

## Environment Diagrams for Nested Def Statements





## 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.

`f1: make_adder`                      `func adder(k) [parent=f1]`



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

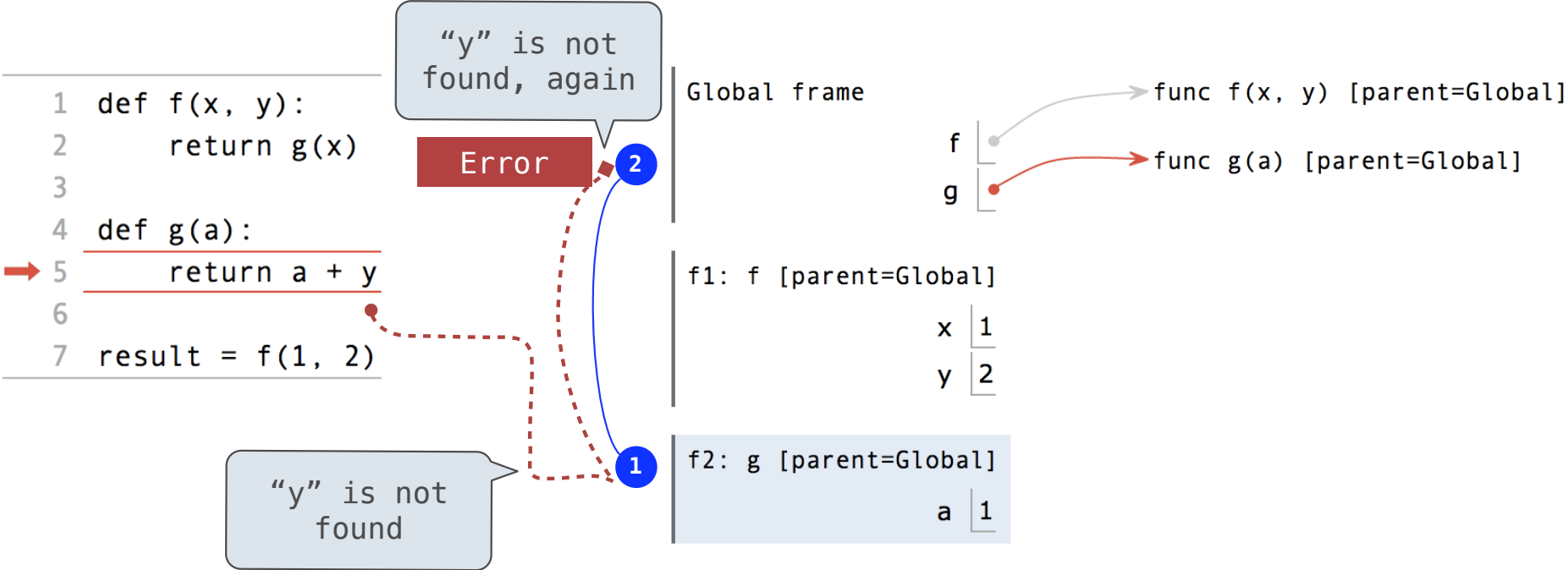
When a function is called:

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.

# Function Composition

(Demo)

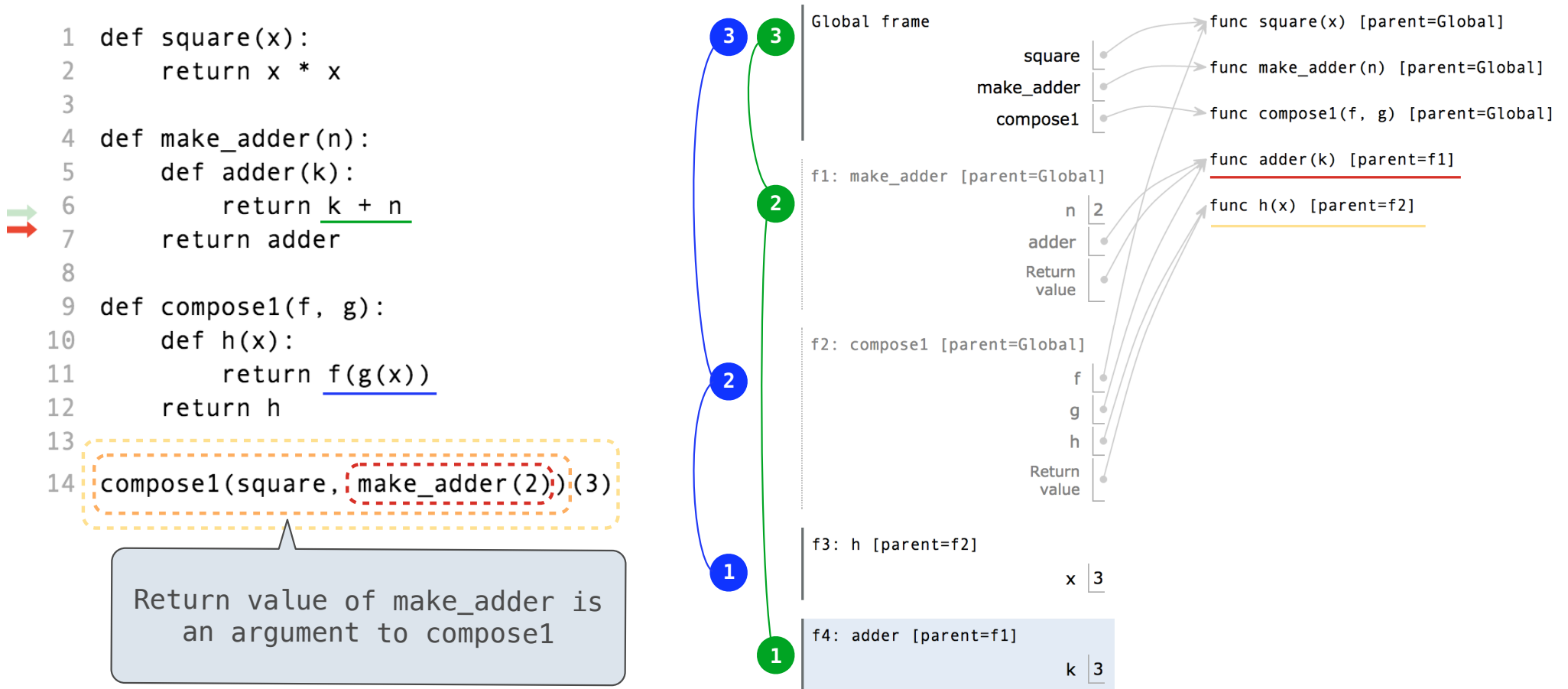
# The Environment Diagram for Function Composition

```

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



# Self-Reference

(Demo)



# Lambda Expressions

(Demo)



## Lambda Expressions

---

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Also an expression: evaluates to a function

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

Important: No "return" keyword!

A function

with formal parameter `x`

that returns the value of `'x * x'`

```
>>> square(4)
16
```

Must be a single expression

Lambda expressions are not common in Python, but important in general  
Lambda expressions in Python cannot contain statements at all!

---

## Lambda Expressions Versus Def Statements



square = lambda x: x \* x

VS



def square(x):  
 return x \* x

- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.

