61A LECTURE 4 – ENVIRONMENTS 2

Steven Tang and Eric Tzeng

June 27, 2013

Announcements

- Homework 1 is due tonight, by 11:59pm!
  - Make sure you leave yourself some time to figure out how submission works!
- Homework 2 is out, due Monday by 11:59
- And expect Homework 3 released sometime this weekend…
- Work on the project!

Congratulations!

- You’ve almost made it through your first week of 61A!
- Just one more day to go!

Higher-Order Functions

Functions are first-class: they can be manipulated as values in Python

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

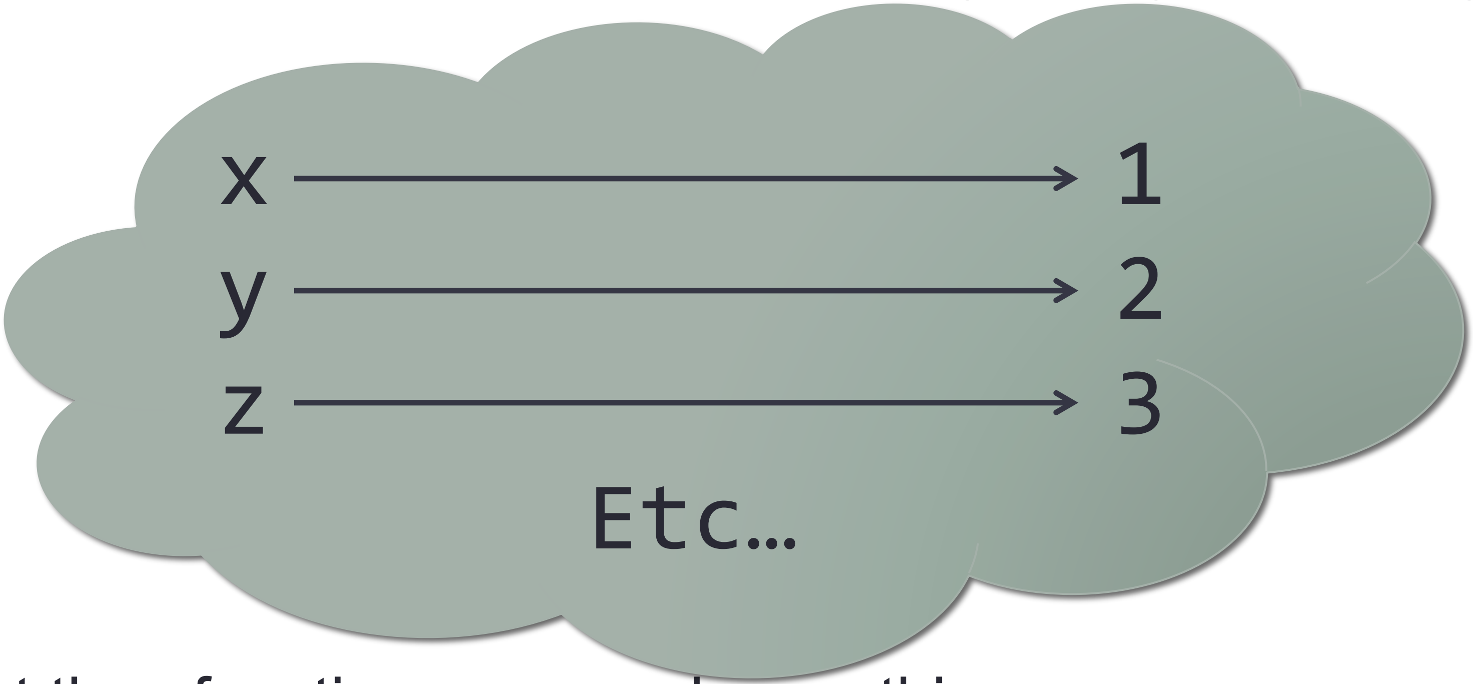
First, some review…

Draw this environment diagram:

```
x = 3

def of_duty()
    return x + 1

def me_maybe(x):
    return of_duty() * x

me_maybe(5)
```

Remember…

- We started off with the idea of having a single mapping:



x → 1

y → 2

z → 3

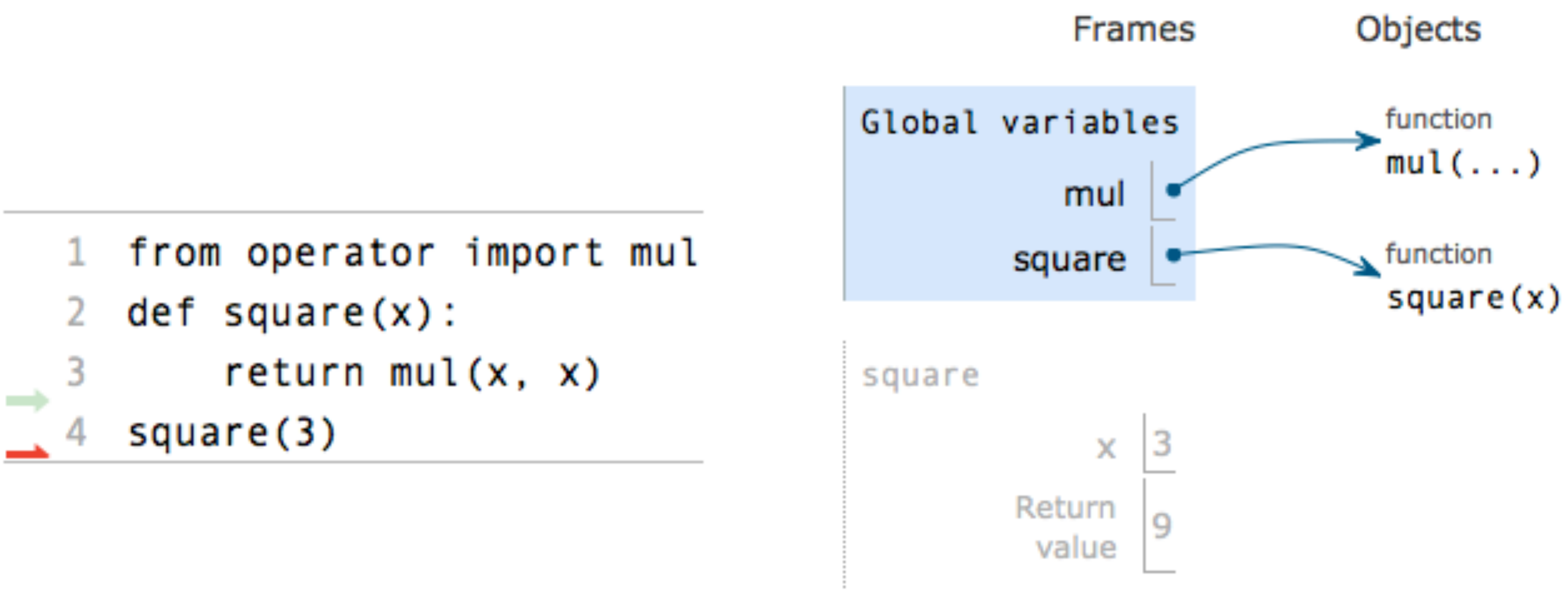
Etc…

- But then functions screwed everything up.

WRONG WRONG WRONG

Remember…

- Then we used environment diagrams (v0.1)…

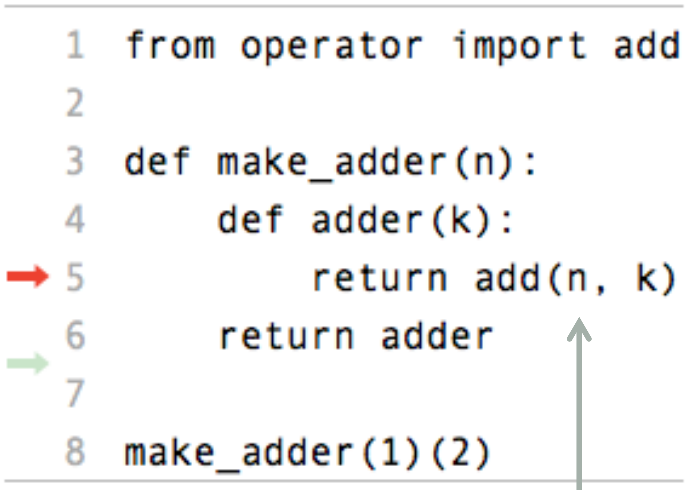


```
from operator import mul
def square(x):
    return mul(x, x)
square(3)
```

- …but even the almighty environment diagram isn't good enough (yet)

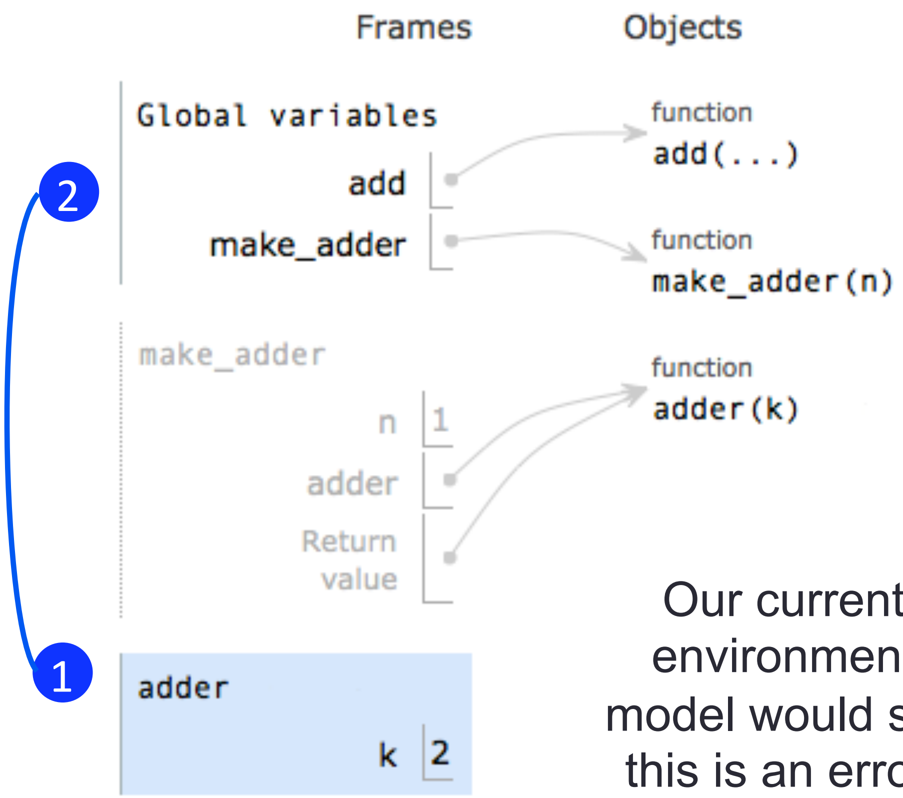
Functions screw everything up again!

- More specifically, higher-order functions!



```
from operator import add

def make_adder(n):
    def adder(k):
        return add(n, k)
    return adder

make_adder(1)(2)
```

Where does n come from?



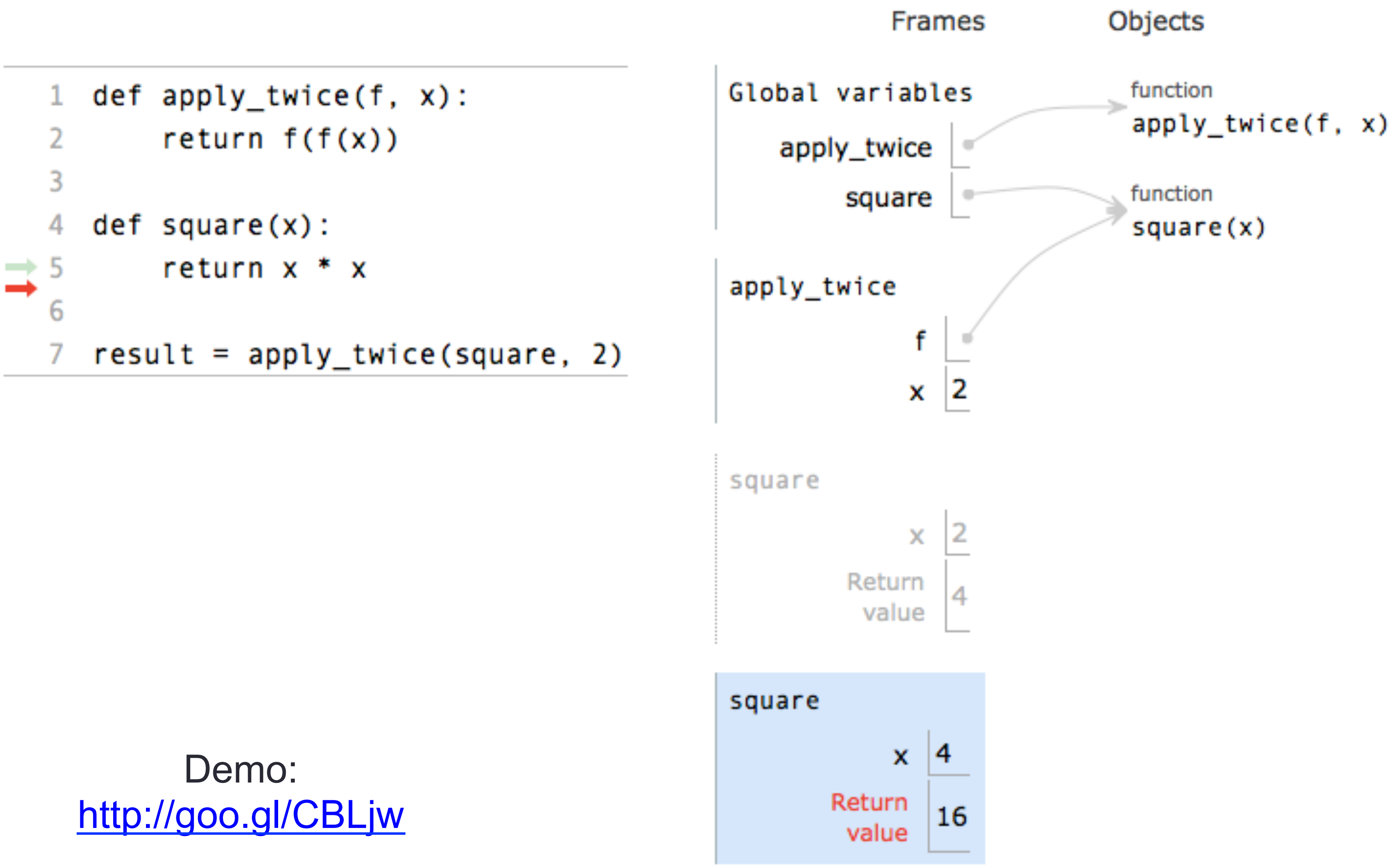
Our current environment model would say this is an error!

- Old model is inadequate

Environments and higher-order functions

- **Higher-order function:** a function that takes a function as an argument value or returns a function as a return value
  - **Functions as arguments:**
    - The environment model we learned already handles that!
    - We'll discuss an example today
  - **Functions as return values:**
    - We need to extend our model a little
    - Change: functions need to know where they were defined
    - Most things stay the same

Functions as arguments



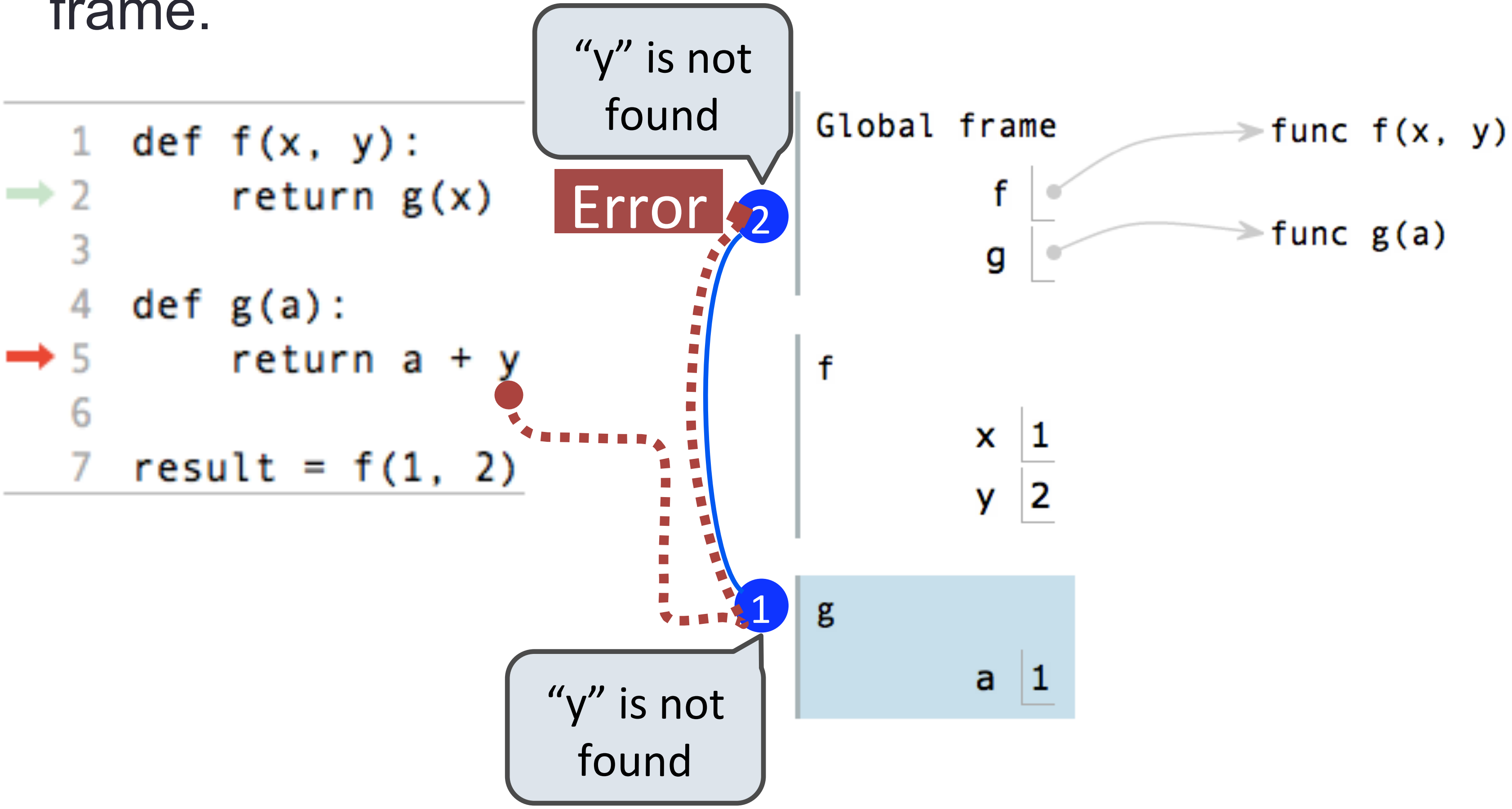
```python
def apply_twice(f, x):
    return f(f(x))

def square(x):
    return x * x

result = apply_twice(square, 2)
```

Demo:
http://goo.gl/CBLjw

Break!

Environments for non-nested functions (review)

- The environment during a call to a non-nested function consists of the newly created local frame and the global frame.

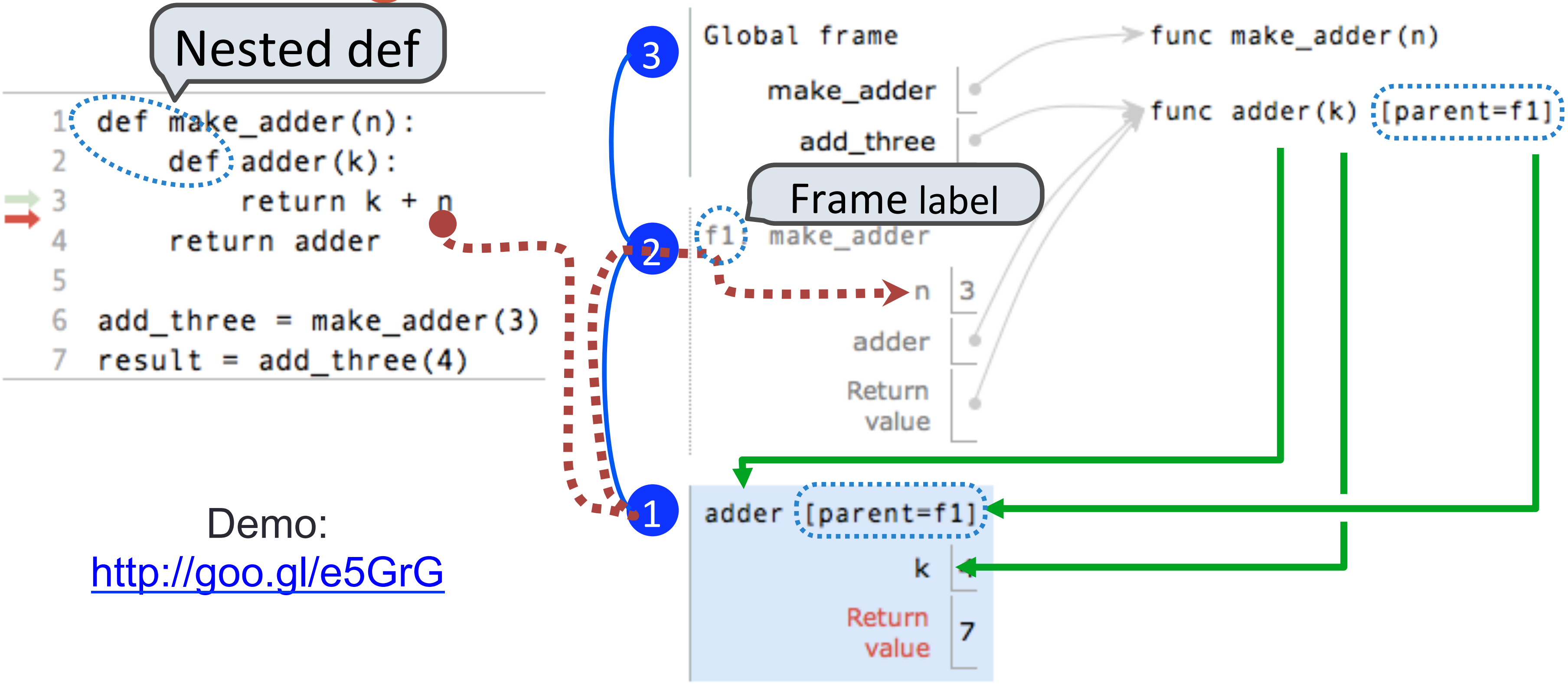


```
def f(x, y):
    return g(x)

def g(a):
    return a + y

result = f(1, 2)
```

What changes with nested functions?

- This is the most important slide of the lecture
- **Before:**
  - The environment during a function call consists of the new local frame and the global frame
  - Check the local frame
  - If not there, check the global frame
- **Now:**
  - The environment during a function call consists of the new local frame and *the environment in which the function was defined*
  - Check the local frame
  - If not there, check the rest of the environment

Env. diagrams for nested functions



Nested def

```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

add_three = make_adder(3)
result = add_three(4)
```

Frame label

Demo:
http://goo.gl/e5GrG

Every user-defined function has a parent frame

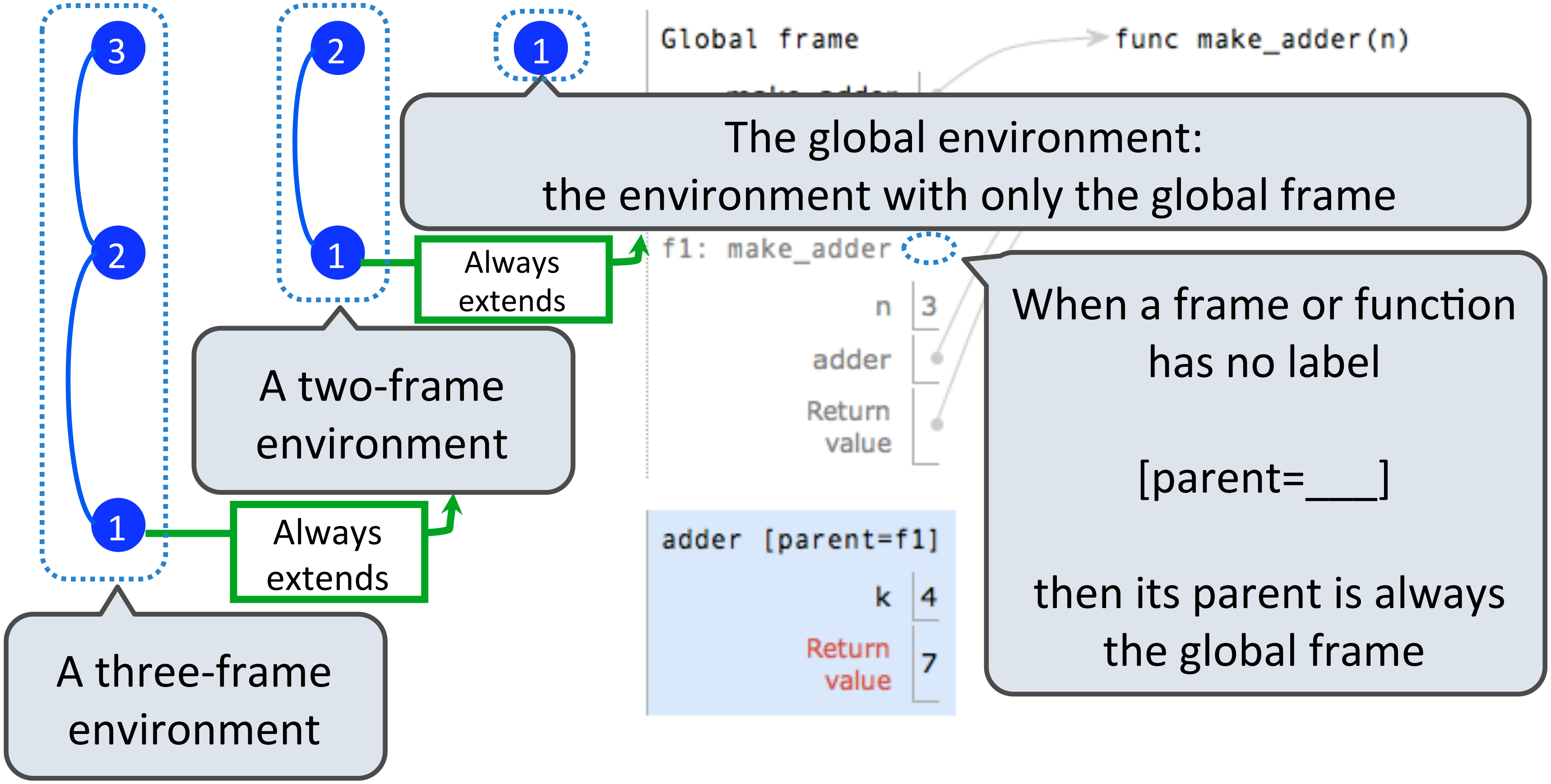
The parent frame of a function is the frame in which it was defined

Every local frame has a parent frame

The parent of a local frame is the parent of the function called

The structure of environments

A frame extends the environment that begins with its parent



```
Global frame                    func make_adder(n)
    make_adder

f1: make_adder
         n  3
     adder
    Return
     value

adder [parent=f1]
         k  4
    Return
     value  7
```

The global environment:
the environment with only the global frame

A two-frame environment

A three-frame environment

When a frame or function has no label

[parent=___]

then its parent is always the global frame

How to draw an environment diagram

When defining a function:

Create a function value with signature
<name>(<formal parameters>)

For nested definitions, label the parent as the first frame of the current environment

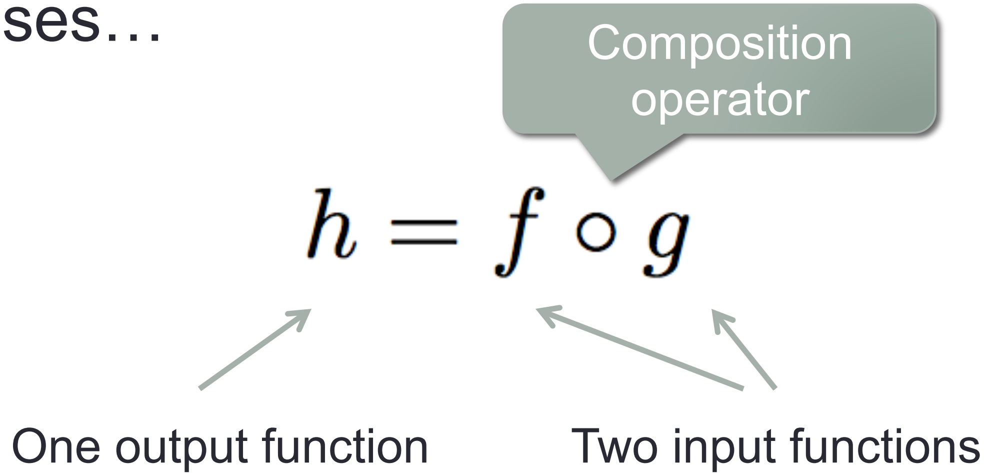
Bind <name> to the function value in the first frame of the current environment

When calling a function:

1. Add a local frame labeled with the <name> of the function
2. If the function has a parent label, copy it to this frame
3. Bind the <formal parameters> to the arguments in this frame
4. Execute the body of the function in the environment that starts with this frame

Example: function composition

- You may be familiar with function composition from your math classes…



Composition operator

$$h = f \circ g$$

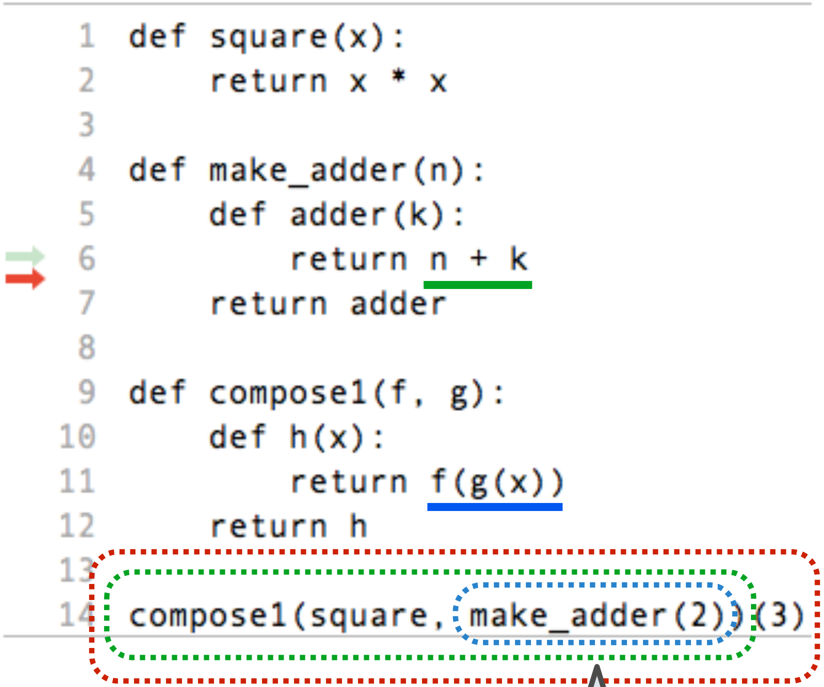
One output function

Two input functions

$$h(x) = f(g(x))$$

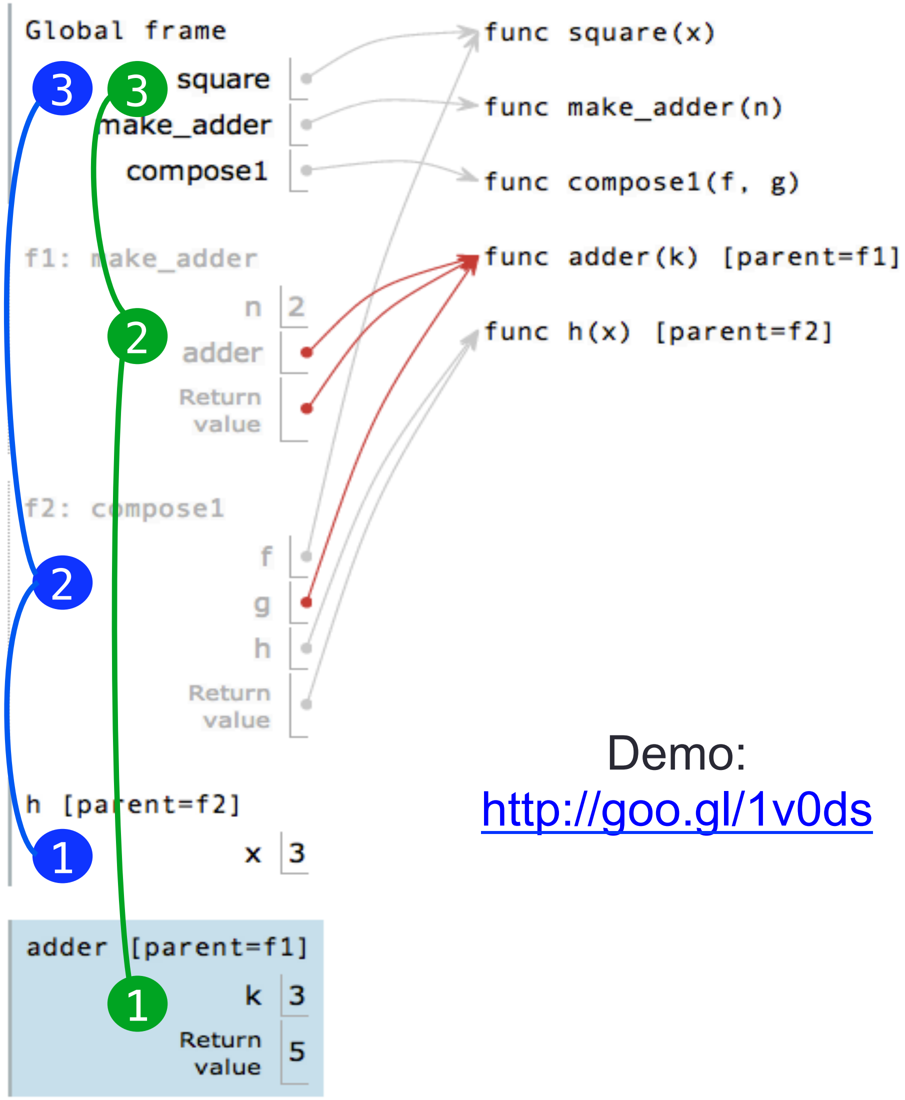
- Code example!

Environment for function composition



```
def square(x):
    return x * x

def make_adder(n):
    def adder(k):
        return n + k
    return adder

def compose1(f, g):
    def h(x):
        return f(g(x))
    return h

compose1(square, make_adder(2))(3)
```

Return value of make_adder is an argument to compose1



Demo: http://goo.gl/1v0ds

Closing remarks…

- We basically only changed one thing: functions now keep an additional bit of information
- With this, your environment model is now complete!
- Practice makes perfect
- Remember it well – if you ever can’t figure out why a variable has a certain value, draw the diagram!