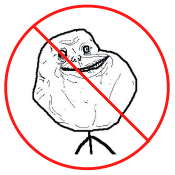
61A Lecture 5

Wednesday, September 5

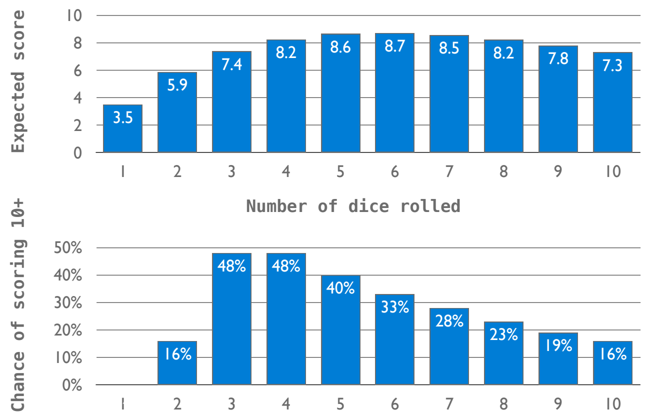
Office Hours: You Should Go!

You are not alone!



http://inst.eecs.berkeley.edu/~cs61a/fa12/staff.html

The Game of Hog



Environments Enable 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:

Our current environment model handles that already!

We'll discuss an example today

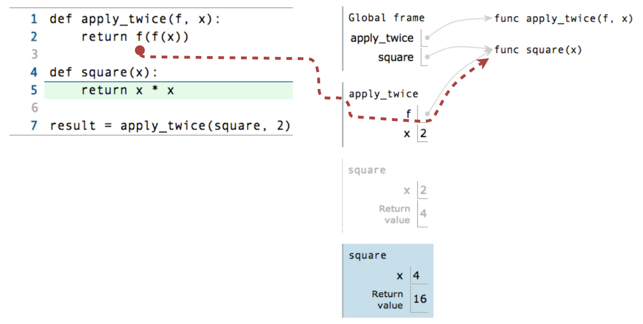
Functions as return values:

We need to extend our model a little

Functions need to know where they were defined

Almost everything stays the same (demo)

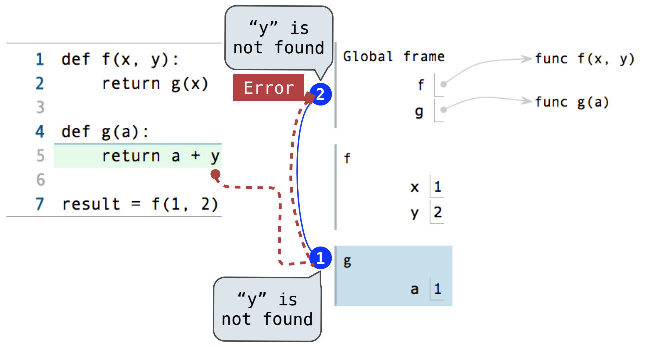
Names Bound to Functional Arguments



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

def square(x):
    return x * x

result = apply_twice(square, 2)
```

Example: http://goo.gl/Gbtc5

Non-Nested Functions Calls Have One Local Frame

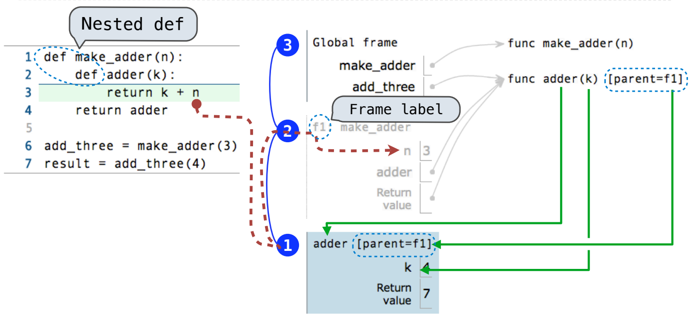


```
def f(x, y):
    return g(x)

def g(a):
    return a + y

result = f(1, 2)
```

- An environment is a sequence of frames
- An environment for a non-nested function (no def within def) consists of one local frame, followed by the global frame

Example: http://goo.gl/tgT5H

Environment Diagrams for Nested Def Statements

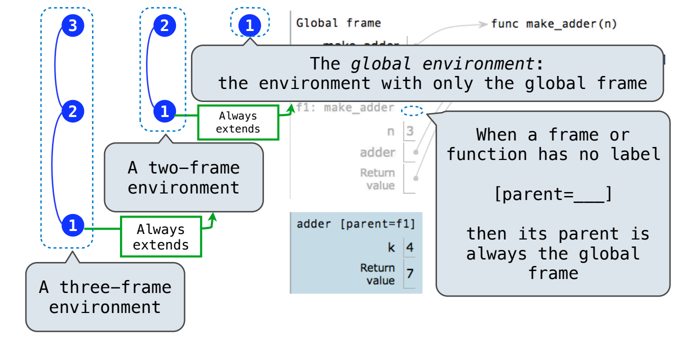


```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

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

- Every user-defined **function** has a *parent frame*
- The parent of a **function** is the frame in which it was *defined*
- Every local **frame** has a *parent frame*
- The parent of a **frame** is the parent of the function *called*

Example: http://goo.gl/L9G2q

The Structure of Environments


3
2
1
2
1
1
Global frame
func make_adder(n)
The *global environment*: the environment with only the global frame
f1: make_adder
n 3
adder
Return value
adder [parent=f1]
k 4
Return value 7
Always extends
A two-frame environment
Always extends
A three-frame environment
When a frame or function has no label [parent=___] then its parent is always the global frame

A frame *extends* the environment that begins with its parent

How to Draw an Environment Diagram

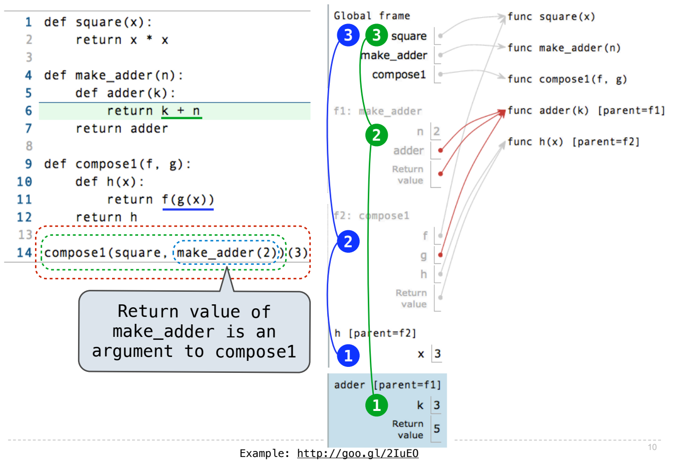
When defining a function:

1. Create a function value with signature *<name>*(*<formal parameters>*)
2. For nested definitions, label the parent as the first frame of the current environment
3. 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

The Environment for Function Composition



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

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

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

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

Example: http://goo.gl/2IuEO

Lambda Expressions

```
>>> ten = 10
```

An expression: this one evaluates to a number

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

Also an expression: evaluates to a function

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

Notice: no "return"

A function
with formal parameter x
and body "return x * x"

Must be a single expression

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

Lambda expressions are rare in Python, but important in general

More Higher-Order Function Examples

(Demo)