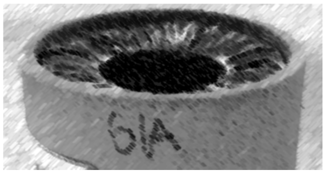
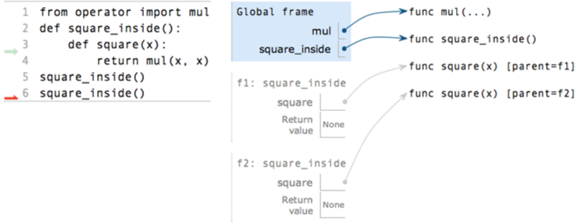


CS61A Lecture 6

Amir Kamil
UC Berkeley
February 4, 2013

Locally Defined Functions

The inner definition is executed each time the outer function is called



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

Example: http://goo.gl/pnU8f

Functions as Return Values

Locally defined functions can be returned

They have access to the frame in which they are defined

```
def make_adder(n):
    """Return a function that adds n to its argument.

    >>> add_three = make_adder(3)
    >>> add_three(4)
    7
    """
    def adder(k):
        return add(n, k)
    return adder
```

A function that returns a function

The name add_three is bound to a function

A local def statement

Can refer to names in the enclosing function

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

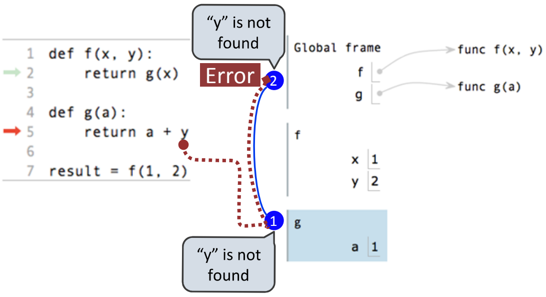
Environment of Function Application

The environment in which a function is applied consists of:

- □ A *new* local frame *each* time the function is *applied*
- □ The environment in which the function was *defined*
  - Before, this was just the global frame
  - For a locally-defined function, this includes all local frames in the definition environment, plus the global frame

Environment for Non-Nested Function

The environment of a function application is a new local frame plus the environment in which the function was *defined*



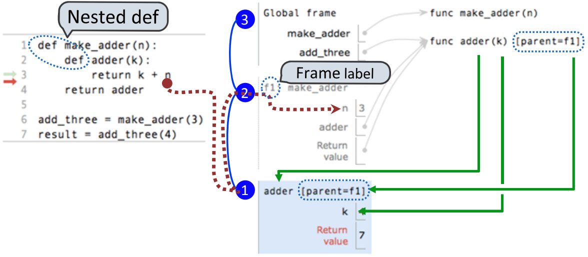
```
def f(x, y):
    return g(x)

def g(a):
    return a + y

result = f(1, 2)
```

Example: http://goo.gl/73anC

Environment for Nested Function



```
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

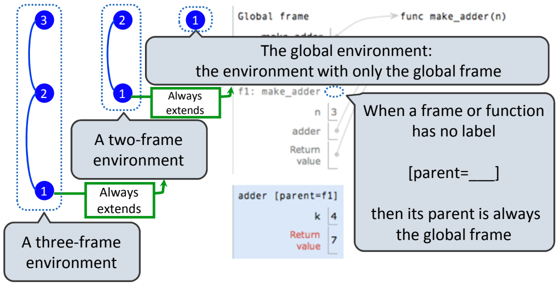
The parent frame of a function is the frame in which it was defined

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



How to Draw an Environment Diagram

When defining a function:

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

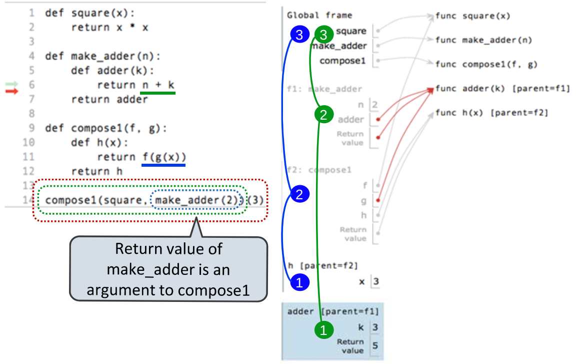
For nested definitions, label the parent as the first frame of the current environment

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

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)
```

Example: http://goo.gl/5zcug

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

Evaluation of Lambda vs. Def

```
lambda x: x * x
```

VS

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

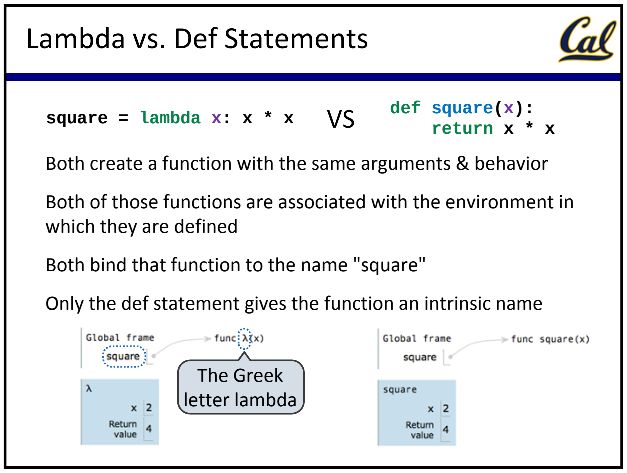
Execution procedure for def *statements*:

1. Create a function value with signature <name>(<formal parameters>) and the current frame as parent
2. Bind <name> to that value in the current frame

Evaluation procedure for lambda *expressions*:

1. Create a function value with signature λ(<formal parameters>) and the current frame as parent
2. Evaluate to that value

No intrinsic name



Lambda vs. Def Statements

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

VS

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

Both create a function with the same arguments & behavior

Both of those functions are associated with the environment in which they are defined

Both bind that function to the name "square"

Only the def statement gives the function an intrinsic name

The Greek letter lambda