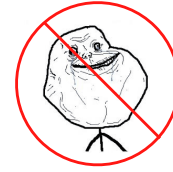


61A Lecture 5

Wednesday, September 7

Office Hours: You Should Go!

You are not alone!



<http://inst.eecs.berkeley.edu/~cs61a/fa11/www/staff.html>

Reminder: Multiple Assignment & Return Values

```
from operator import floordiv, mod

def divide_exact(n, d):
    """Return the quotient and remainder of dividing n by d.

    >>> q, r = divide_exact(13, 5)
    >>> q
    2
    >>> r
    3
    """
    return floordiv(n, d), mod(n, d)
```

Integer division, which rounds down

Integer remainder after dividing

Multiple assignment to two names

Multiple return values, separated by commas

The Structure of Project 1

Two functions implement the game simulation

Warning!
Pseudo-code
(not code)

```
def play(...):
    while game is not over:
        get a plan (from the current player's strategy)
        call take_turn with a dice and plan
    return winner

def take_turn(...):
    while turn is not over:
        get an action (from plan) and outcome (from dice)
        call an action
    return points scored during the turn
```

The Structure of Project 1

Four types of functions are involved in simulating game

	Domain	Range
Action	(integer, integer) Two arguments	(integer, integer, boolean) Three return values
Plan	integer	Action
Strategy	(integer, integer)	Plan
Dice	No arguments	integer

The Purpose of 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

Review: Summation Example

```
def cube(k):
    return pow(k, 3)
```

Function of a single argument (not called term)

```
def summation(n, term):
    """Sum the first n terms of a sequence.
```

A formal parameter that will be bound to a function

```
>>> summation(5, cube)
```

```
225
```

```
total, k = 0, 1
while k <= n:
    total, k = total + term(k), k + 1
return total
```

The cube function is passed as an argument value

The function bound to term gets called here

$0 + 1^3 + 2^3 + 3^3 + 4^3 + 5^3$

Environments Enable Higher-Order Functions!

Functions as arguments:

Our current environment model handles that!

We'll give an example of how

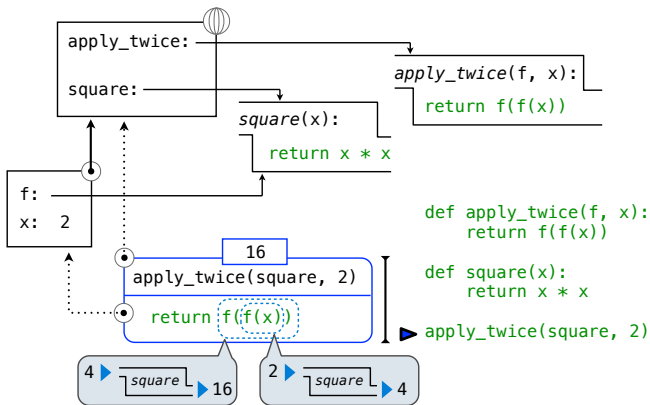
Functions as return values:

We need to extend the model a little

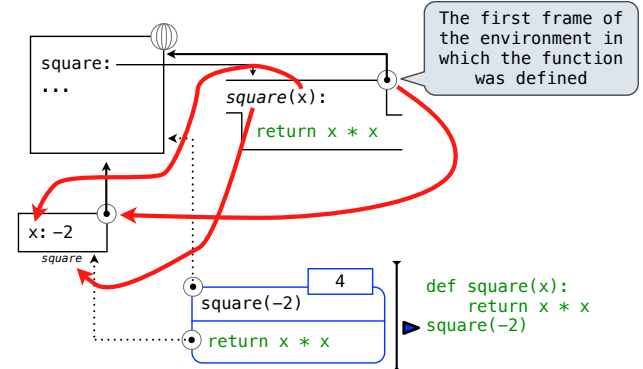
Functions need to know where they were defined

Almost everything stays the same

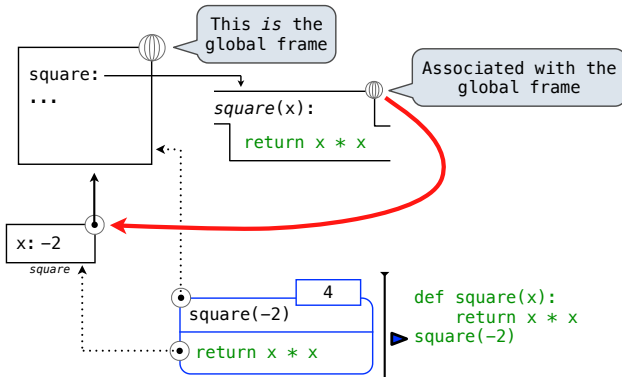
Names and Environments with Functional Values



Applying User-Defined Functions



Functions Associated with the Global Frame



Locally Defined Functions: Example

Functions defined within other function bodies are bound to names in the local frame

A function that returns a function

```
def make_adder(n):
    """Return a function that takes one argument k and returns k + n.
    """
    >>> (add_three = make_adder(3))
    >>> add_three(4)
    7
```

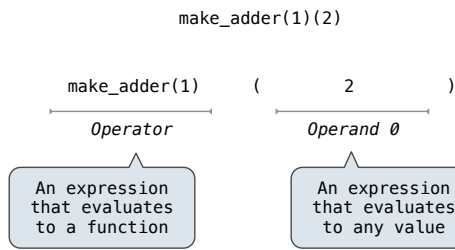
The name add_three is bound to a function

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

A local def statement

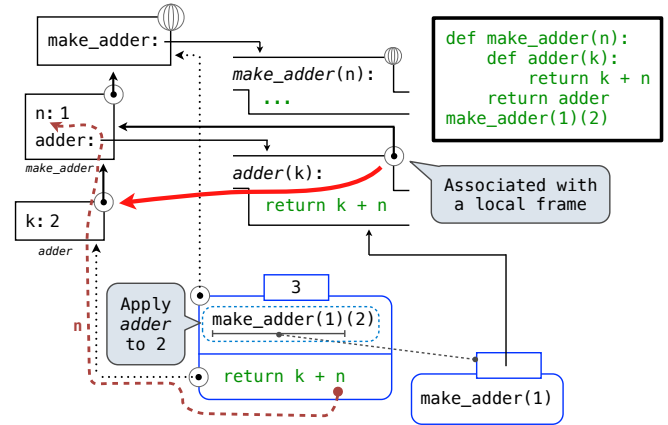
Can refer to names in the enclosing function

Locally Defined Functions: Call Expressions

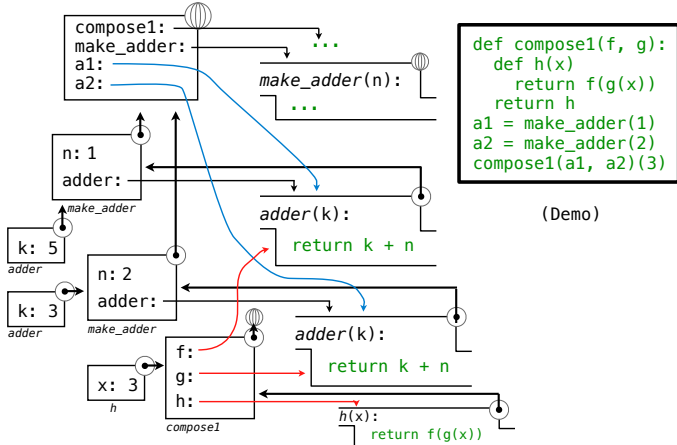


```
def make_adder(n):
    def adder(k):
        return k + n
    return adder
make_adder(1)(2)
```

Locally Defined Functions: Environments



The Environment for Function Composition



Lambda Expressions

```
>>> ten = 10
>>> square = x * x
>>> square = lambda x: x * x
>>> square(4)
16
```

An expression: this one evaluates to a number

Also an expression: evaluates to a function

Notice: no "return"

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

Must be a single expression

Lambda expressions are rare in Python, but important in general

More Higher-Order Function Examples

(Demo)