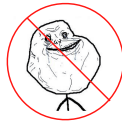


61A Lecture 5

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

- func apply_twice(f, x) [parent=Global]
- apply_twice
- square

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)
```

Global frame

- func apply_twice(f, x) [parent=Global]
- apply_twice
- square
- f1: apply_twice [parent=Global]
- x: 2

Environments for Nested Definitions

(Demo)

Environment Diagrams for Nested Def Statements

```
1 def make_adder(n):  
2   def adder(k):  
3     return k + n  
4   return adder  
5  
6 add_three = make_adder(3)  
7 add_three(4)
```

Nested def

Global frame

- func make_adder(n) [parent=Global]
- make_adder
- func adder(k) [parent=f1]
- f1: make_adder [parent=Global]
- adder
- Return value
- f2: adder [parent=f1]
- k: 4
- Return value: 7

- Every user-defined function has a parent frame (often global)
- The parent of a function is the frame in which it was defined
- Every local frame has a parent frame (often global)
- The parent of a frame is the parent of the function called

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

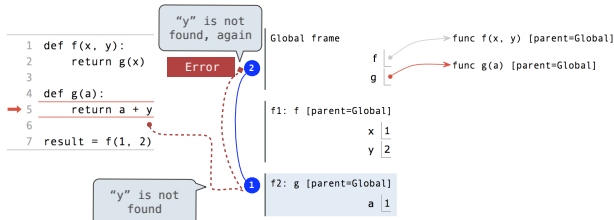
When a function is called:

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

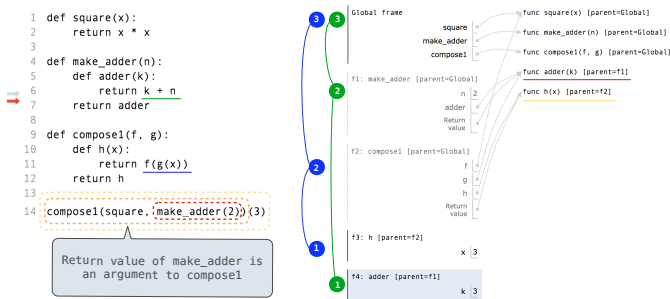


- An environment is a sequence of frames.
- The environment created by calling a top-level function (no def within def) consists of one local frame, followed by the global frame.

Function Composition

(Demo)

The Environment Diagram for Function Composition



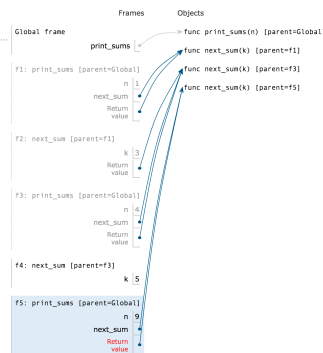
Self-Reference

(Demo)

Returning a Function by Its Own Name

```

1 def print_sums(n):
2   print(n)
3   def next_sum(k):
4     return print_sums(n+k)
5   return next_sum
6
7 print_sums(1)(3)(5)
    
```



Lambda Expressions

(Demo)

Lambda Expressions

```
>>> x = 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



Important: No "return" keyword!

A function with formal parameter x that returns the value of 'x * x'

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.

