Environments

Announcements

Environments for Higher-Order Functions

Environments Enable Higher-Order Functions

Functions are first-class: Functions are values in our programming language

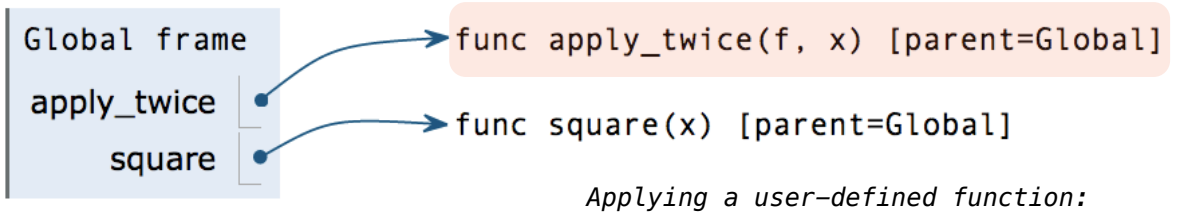
Higher-order function: A function that takes a function as an argument value **or**
A function that returns a function as a return value

Environment diagrams describe how higher-order functions work!

(Demo)

Names can be Bound to Functional Arguments

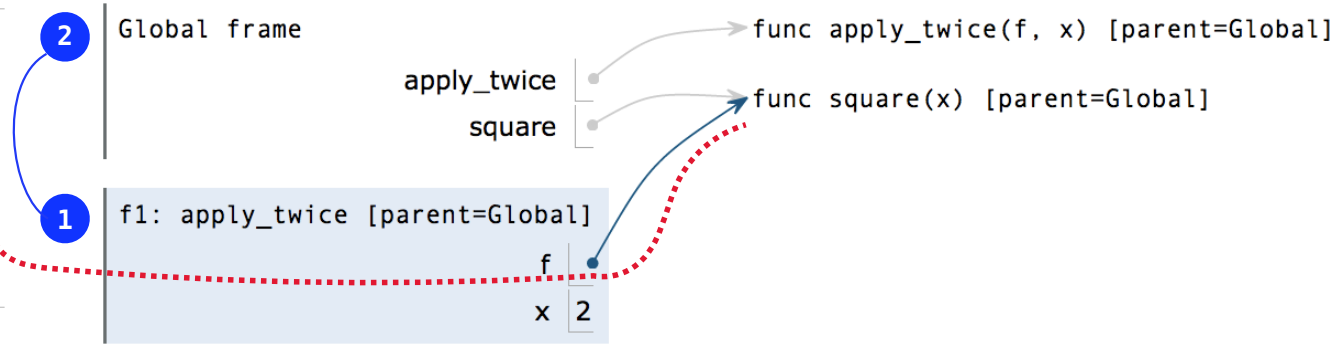
```
1  def apply_twice(f, x):
2      return f(f(x))
3
4  def square(x):
5      return x * x
6
7  result = apply_twice(square, 2)
```



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)
```



pythontutor.com/composingprograms.html#code=def%20apply_twice%28f,%20x%29%3A%0A%20%20%20%20return%20f%28f%28x%29%29%0A%20%20%20%20%0Adef%20square%28x%29%3A%0A%20%20%20%20return%20x%20*%20x%0A%20%20%20%20%0Aresult%20%3D%20apply_twice%28square,%202%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

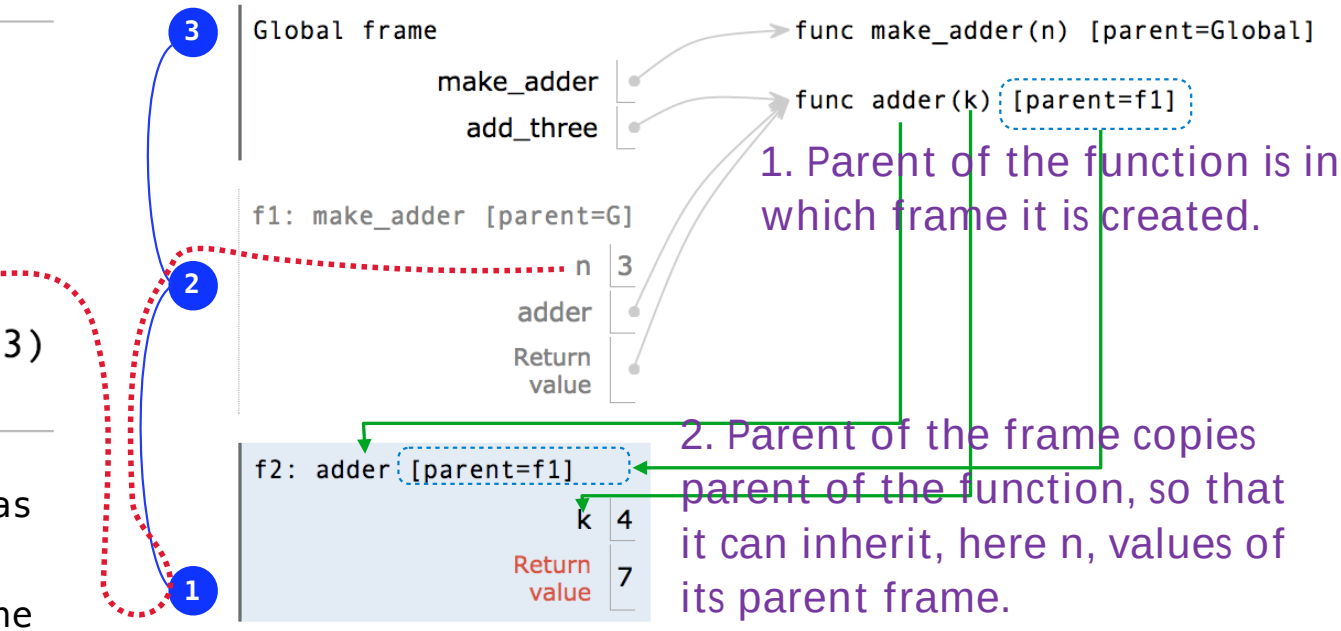
Environments for Nested Definitions

(Demo)

Environment Diagrams for Nested Def Statements

Nested def

```
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)
```



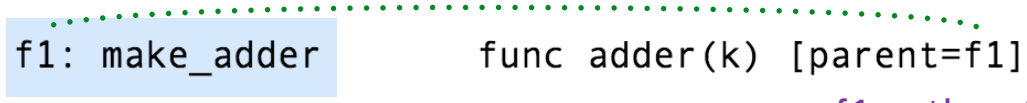
- Every user-defined function has a parent frame (often global)
- The parent of a function is the frame in which it was defined
- Every local frame has a parent frame (often global)
- The parent of a frame is the parent of the function called

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.



Bind <name> to the function value in the current frame

use f1 rather than make_adder because there might be many frames when calling make_adder several times

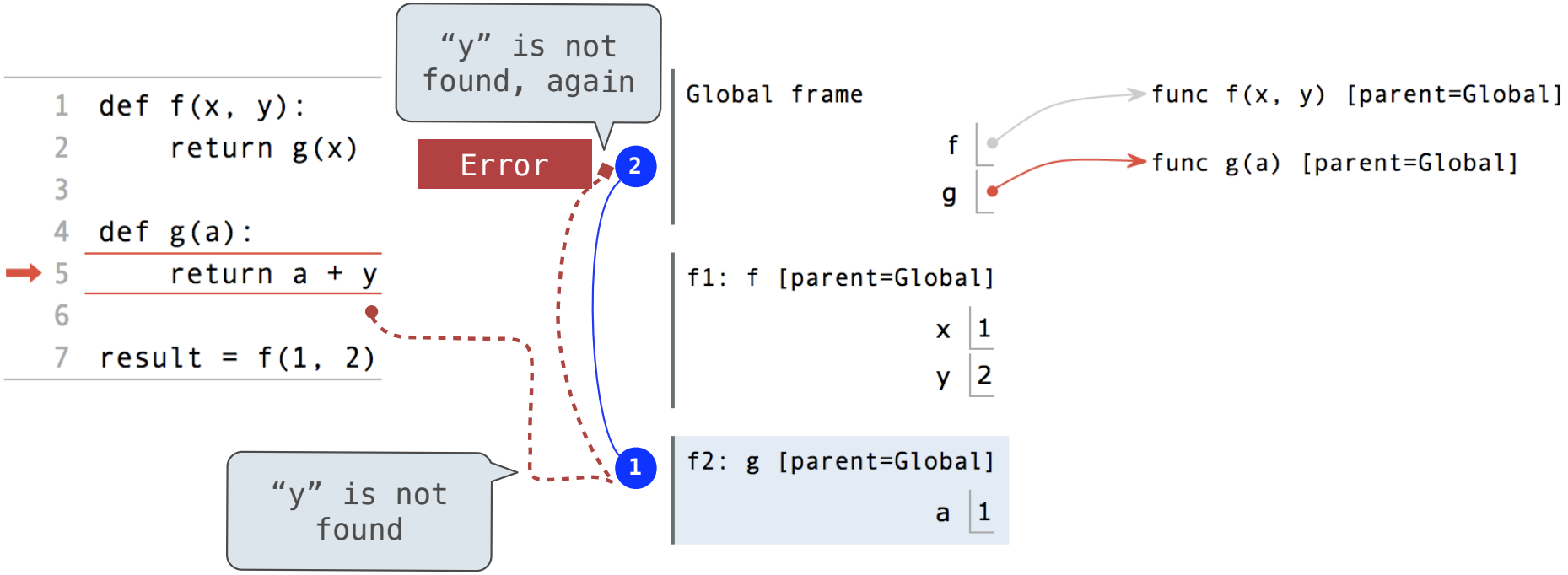
When a function is called:

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.

http://pythontutor.com/composingprograms.html#code=def%20f%28x,
%20y%29%3A%0A%20%20%20%20return%20g%28x%29%0A%0Adef%20g%28a%29%3A%0A%20%20%20%20return%20a%20%2B%20y%0A%20%20%20%20%0Aresult%20%3D%20f%281,%202%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

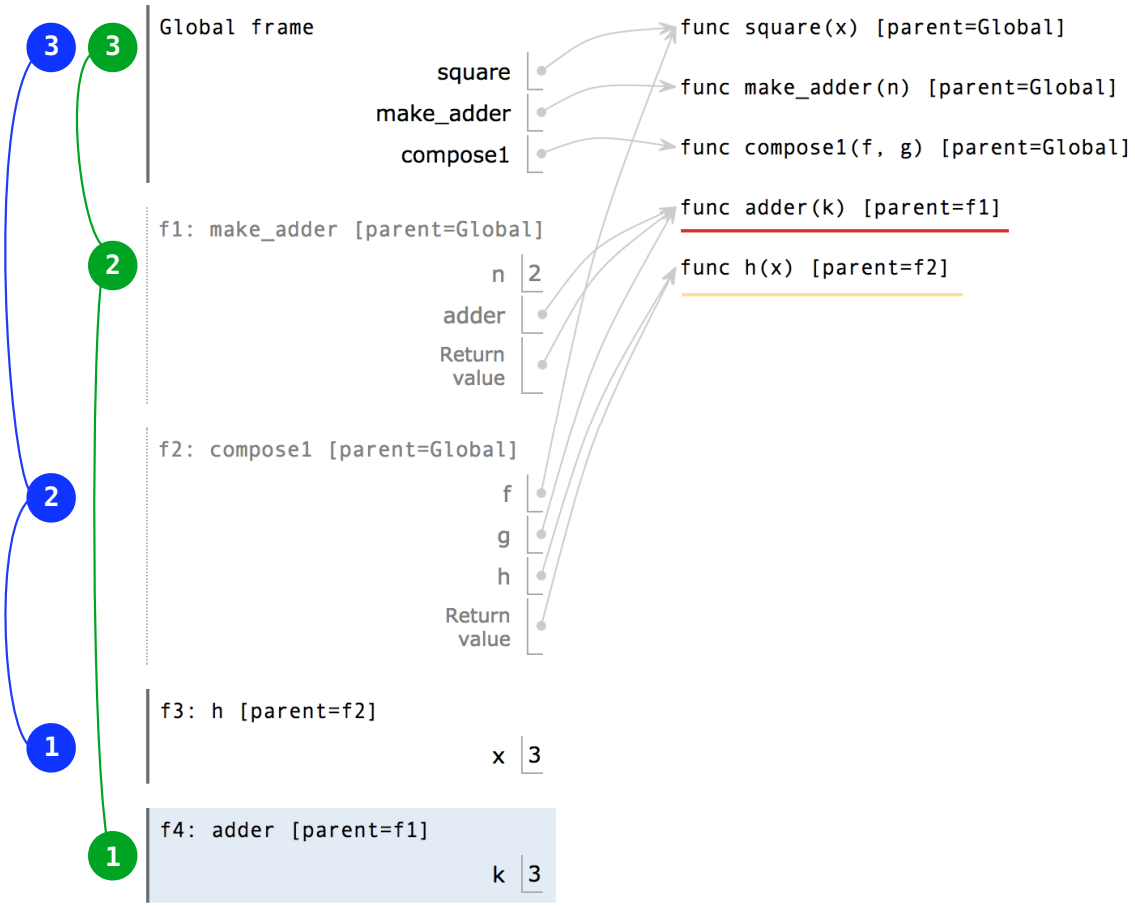
Function Composition

(Demo)

The Environment Diagram for Function Composition

```
1  def square(x):
2      return x * x
3
4  def make_adder(n):
5      def adder(k):
6          return k + n
7      return adder
8
9  def compose1(f, g):
10     def h(x):
11         return f(g(x))
12     return h
13
14 compose1(square, make_adder(2))(3)
```

Return value of make_adder is an argument to compose1

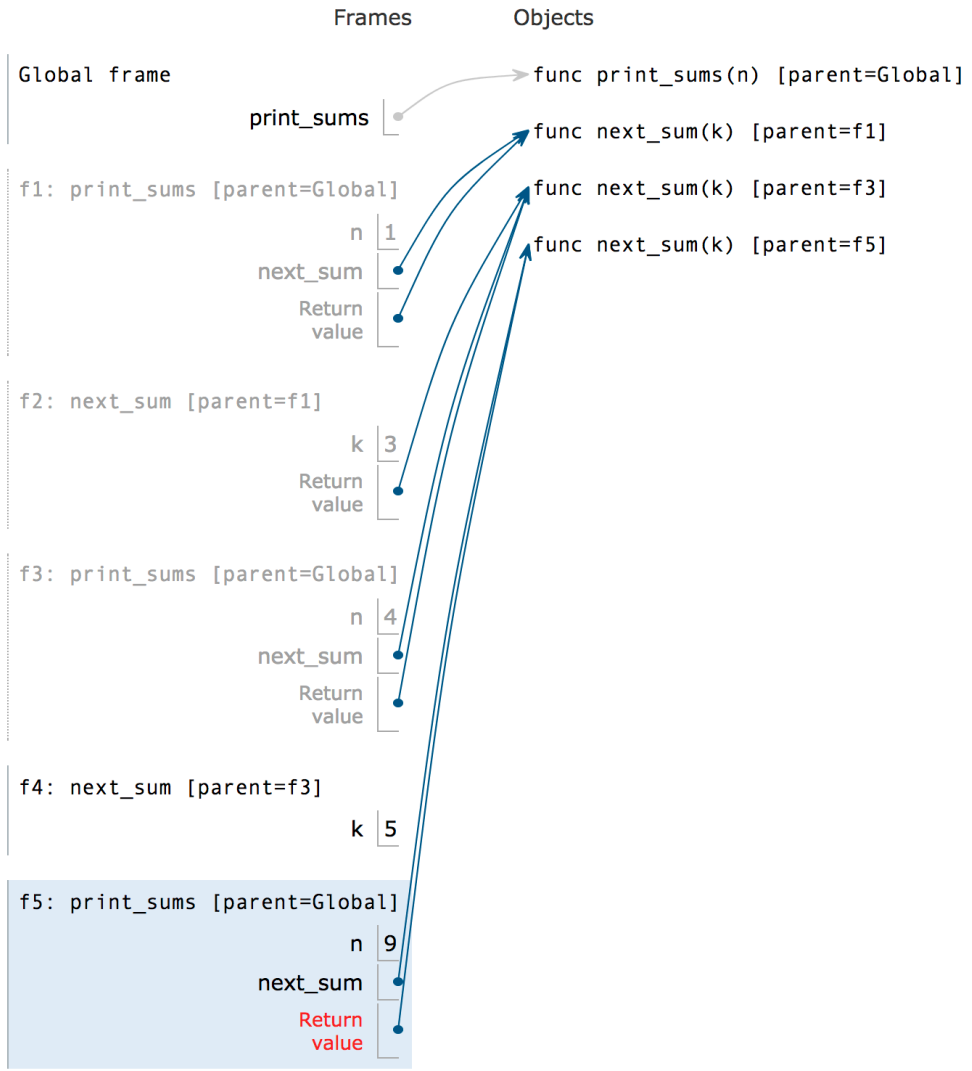


Self-Reference

(Demo)

Returning a Function Using 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)
```



http://pythontutor.com/composingprograms.html#code=def%20print_all%28k%29%3A%0A%20%20%20%20print%28k%29%0A%20%20%20%20return%20print_all%0A%20%20%20%20%0Aprint_all%281%29%283%29%285%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

http://pythontutor.com/composingprograms.html#code=def%20print_sums%28n%29%3A%0A%20%20%20%20print%28n%29%0A%20%20%20%20def%20next_sum%28k%29%3A%0A%20%20%20%20%20%20%20%20return%20print_sums%28n%2Bk%29%0A%20%20%20%20return%20next_sum%0A%0Aprint_sums%281%29%283%29%285%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D