**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**

Environments for Nested Definitions

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.

Fl: make_adder func adder(k) [parent=fl]

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

Local Names are not Visible to Other (Non-Nested) Functions

- "y" is not found
- "y" is not found, again

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.

Lambda Expressions

An expression: this one evaluates to a number
- Also an expression: evaluates to a function
  - that returns the value of "x * x"
- A function with formal parameter x
- A function

Lambda expressions are not common in Python, but important in general
- Important: No "return" keyword!
- Must be a single expression

Lambda expressions in Python cannot contain statements at all!

Lambda Expressions Versus Def Statements

- Both create a function with the same domain, range, and behavior.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name, which shows up in
  environment diagrams but doesn't affect execution (unless the function is printed).

Function Composition

The Environment Diagram for Function Composition

Self-Reference
Function Currying

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

>>> make_adder(2)(3)
5
>>> add(2, 3)
5

Curry: Transform a multi-argument function into a single-argument, higher-order function