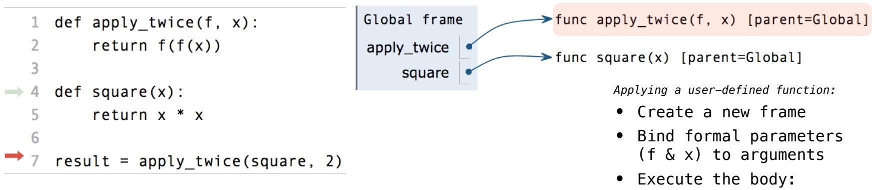
Higher-Order Environments

Announcements

Environments for Higher-Order Functions

(Demo)

Names can be Bound to Functional Arguments

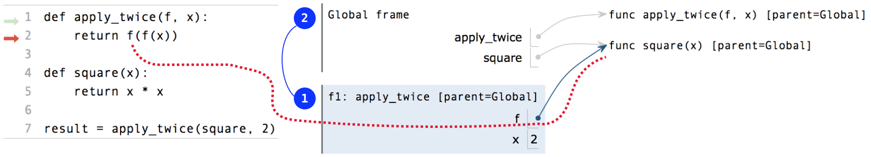


```
def apply_twice(f, x):
    return f(f(x))

def square(x):
    return x * x

result = apply_twice(square, 2)
```

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

Applying a user-defined function:

- Create a new frame
- Bind formal parameters (f & x) to arguments
- Execute the body: return f(f(x))



```
def apply_twice(f, x):
    return f(f(x))

def square(x):
    return x * x

result = apply_twice(square, 2)
```

2
1
Global frame
apply_twice
square
func apply_twice(f, x) [parent=Global]
func square(x) [parent=Global]
f1: apply_twice [parent=Global]
f
x 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

Types of Higher-Order Functions

Environments Enable Higher-Order Functions

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

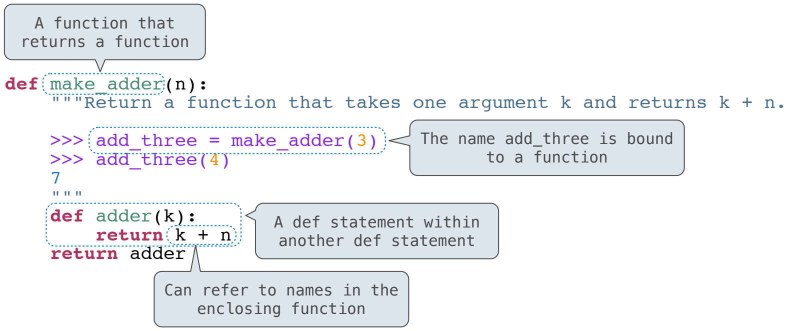
Higher-order function: A function that takes a function as an argument value **or**
A function that returns a function as a return value

(Demo)

Functions as Return Values

Locally Defined Functions

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



A function that returns a function

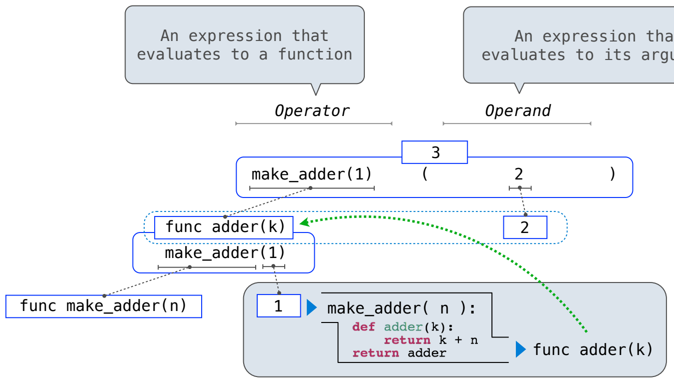
```
def make_adder(n):
    """Return a function that takes one argument k and returns k + n.

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

The name add_three is bound to a function

A def statement within another def statement

Can refer to names in the enclosing function

Call Expressions as Operator Expressions



An expression that evaluates to a function

An expression that evaluates to its argument

Operator

Operand

3

make_adder(1) (2)

func adder(k)

2

make_adder(1)

func make_adder(n)

1

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

func adder(k)

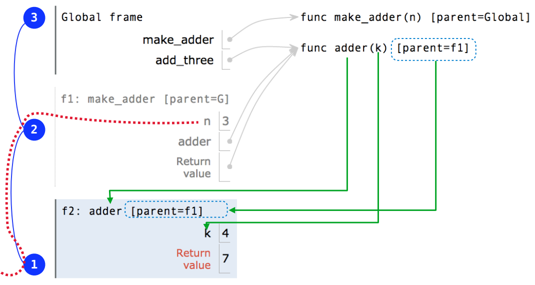
Environments for Nested Definitions

(Demo)

Environment Diagrams for Nested Def Statements

Nested def

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

add_three = make_adder(3)
add_three(4)
```



Global frame

make_adder

add_three

func make_adder(n) [parent=Global]

func adder(k) [parent=f1]

f1: make_adder [parent=G]

n 3

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

http://pythontutor.com/composingprograms.html#code=def%20make_adder%28n%29%3A%0A%20%20%20%20def%20adder%28k%29%3A%0A%20%20%20%20%20%20%20%20return%20k%20%2B%20n%0A%20%20%20%20return%20adder%0A%20%20%20%20%0Athree_more_than%20%3D%20make_adder%283%29%0Aresult%20%3D%20three_more_than%284%29&cumulative=false&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

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.

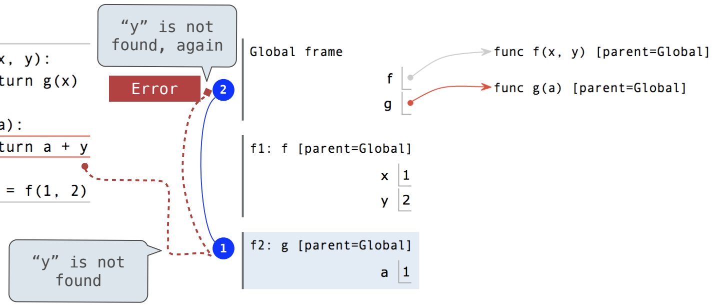
4. Execute the body of the function in the environment that starts with the local frame.

Local Names

(Demo)

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

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

def g(a):
    return a + y

result = f(1, 2)
```



- 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

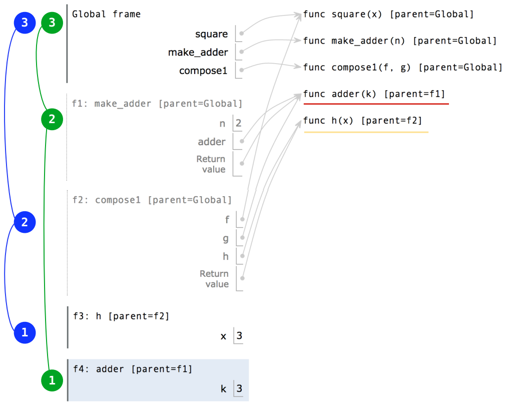
Function Composition

(Demo)

The Environment Diagram for Function Composition

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

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

def compose1(f, g):
    def h(x):
        return f(g(x))
    return h

compose1(square, make_adder(2))(3)
```

Return value of make_adder is an argument to compose1



3
3
Global frame
square
make_adder
compose1
func square(x) [parent=Global]
func make_adder(n) [parent=Global]
func compose1(f, g) [parent=Global]
func adder(k) [parent=f1]
func h(x) [parent=f2]
2
f1: make_adder [parent=Global]
n 2
adder
Return value
2
f2: compose1 [parent=Global]
f
g
h
Return value
1
f3: h [parent=f2]
x 3
1
f4: adder [parent=f1]
k 3

http://pythontutor.com/composingprograms.html#code=def%20square%28x%29%3A%0A%20%20%20%20return%20x%20*%20x%0A%0Adef%20make_adder%28n%29%3A%0A%20%20%20%20def%20adder%28k%29%3A%0A%20%20%20%20%20%20%20%20return%20k%20%2B%20n%0A%20%20%20%20return%20adder%0A%20%20%20%20%0Adef%20compose1%28f,
%20g%29%3A%0A%20%20%20%20def%20h%28x%29%3A%0A%20%20%20%20%20%20%20%20return%20f%28g%28x%29%29%0A%20%20%20%20return%20h%0A%20%20%20%20%0Acompose1%28square,%20make_adder%282%29%29%283%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

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!

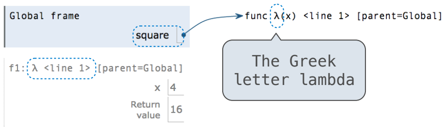
Lambda Expressions Versus Def Statements

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

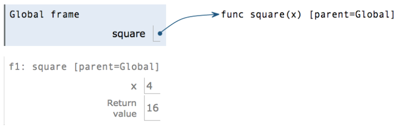
VS

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

- Both create a function with the same domain, range, and behavior.
- Both 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).


Global frame
square
func λ(x) <line 1> [parent=Global]
f1: λ <line 1> [parent=Global]
x 4
Return value 16

The Greek letter lambda


Global frame
square
func square(x) [parent=Global]
f1: square [parent=Global]
x 4
Return value 16