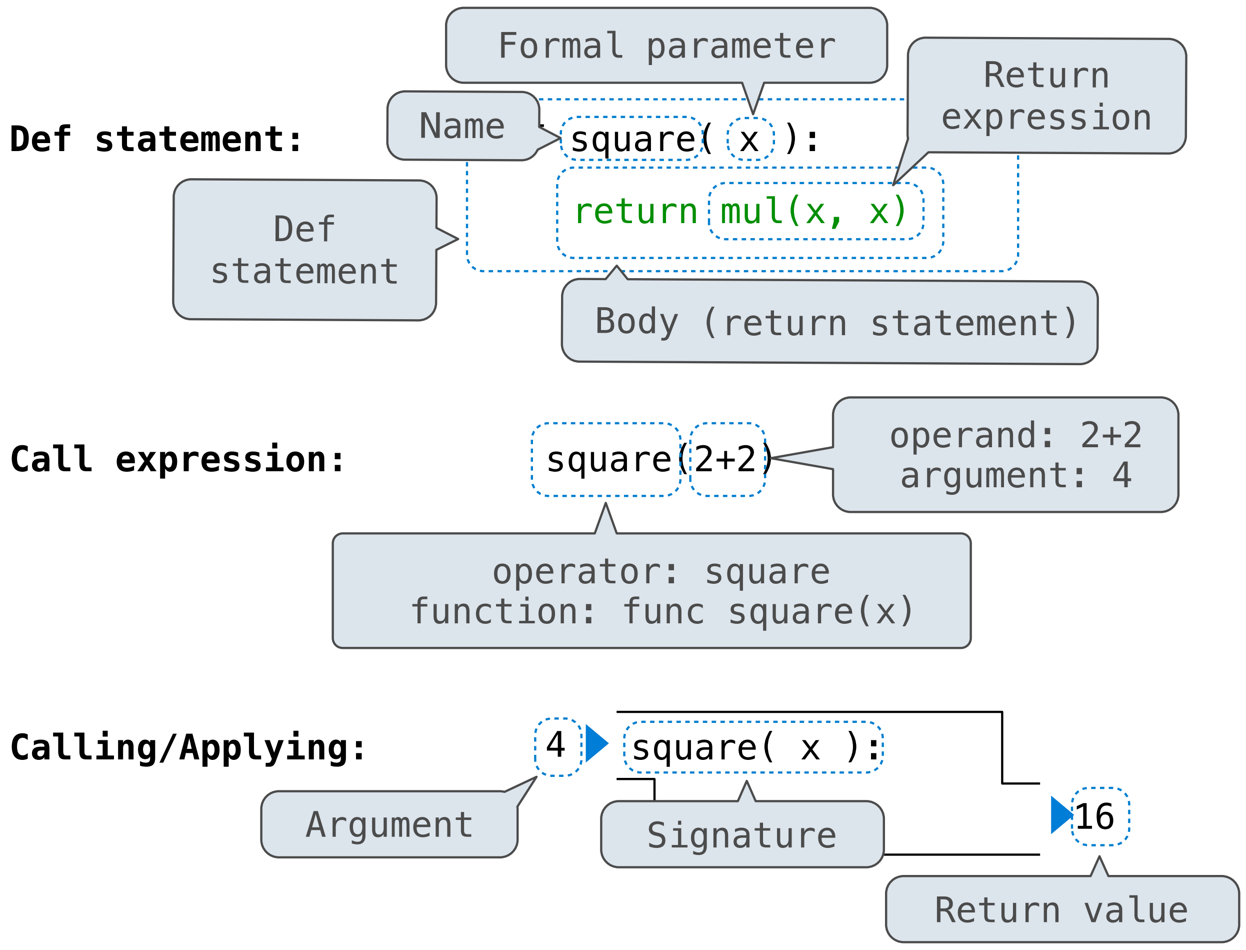
Multiple Environments

Life Cycle of a User-Defined Function



What happens?

A new function is created!

Name bound to that function in the current frame

Operator & operands evaluated

Function (value of operator) called on arguments (values of operands)

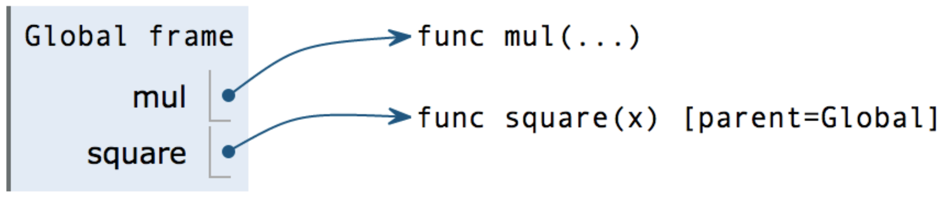
A new frame is created!

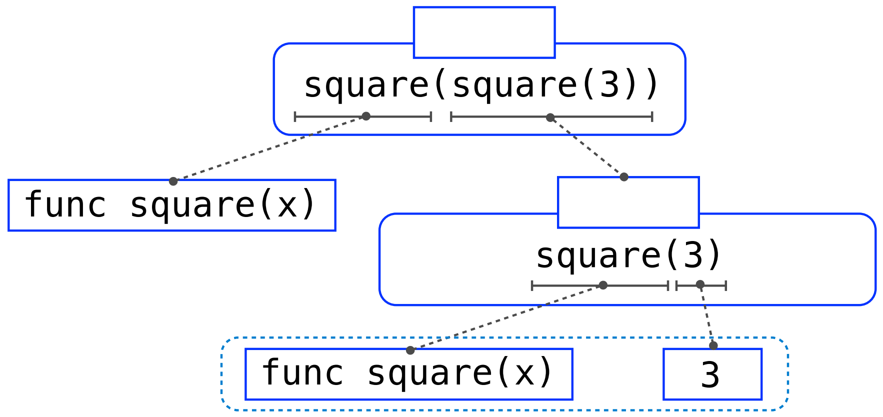
Parameters bound to arguments

Body is executed in that new environment

Multiple Environments in One Diagram!

```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

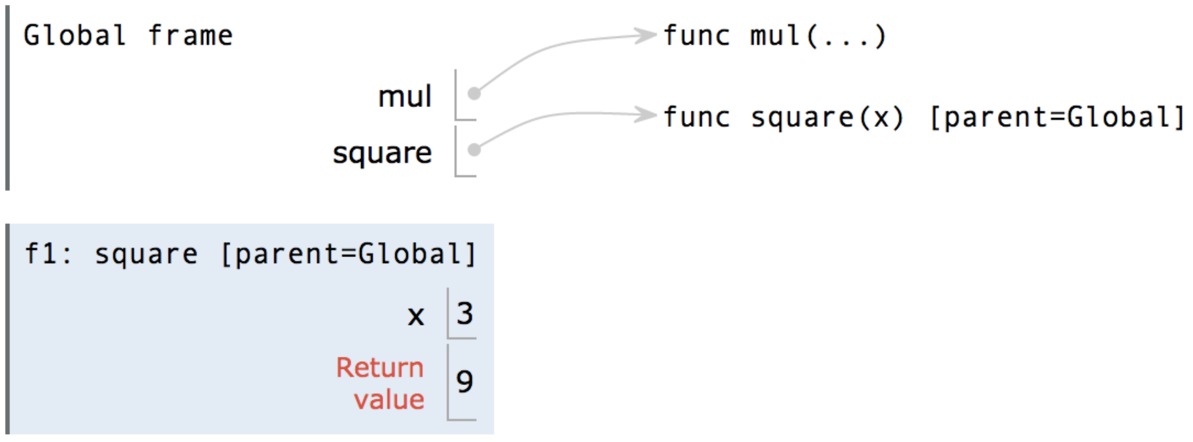


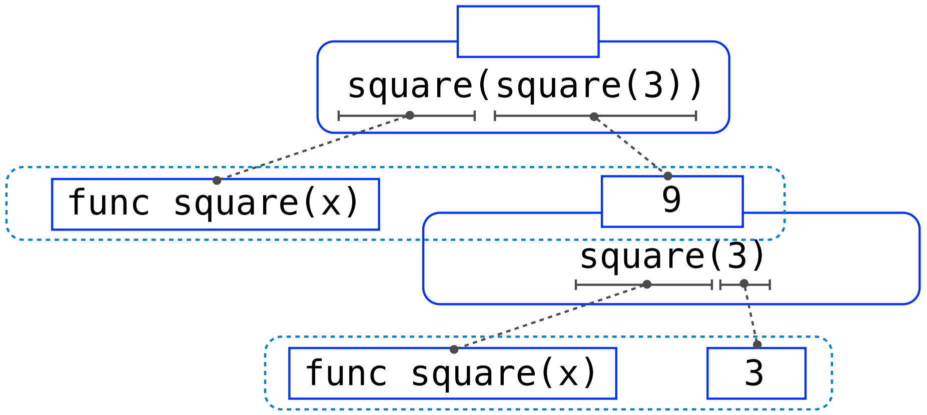


Interactive Diagram

Multiple Environments in One Diagram!

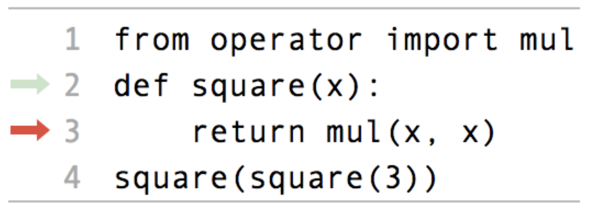
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



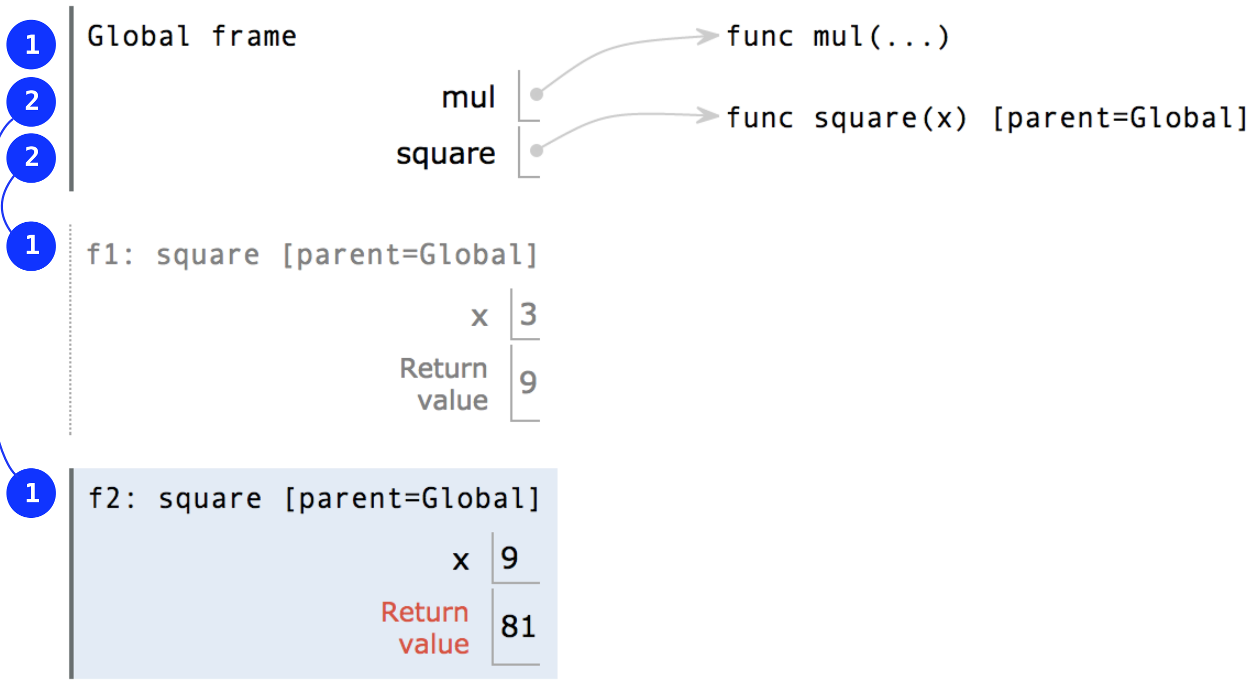


Interactive Diagram

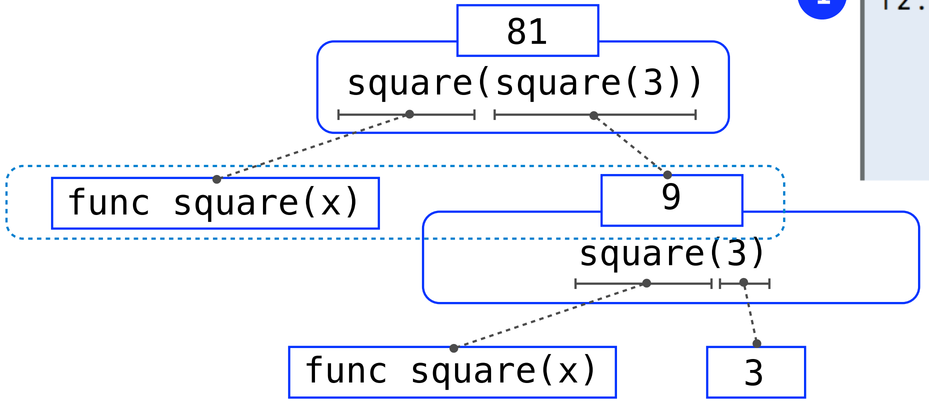
Multiple Environments in One Diagram!



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



1
2
2
1
1
Global frame
mul
square
func mul(...)
func square(x) [parent=Global]
f1: square [parent=Global]
x
3
Return value
9
f2: square [parent=Global]
x
9
Return value
81

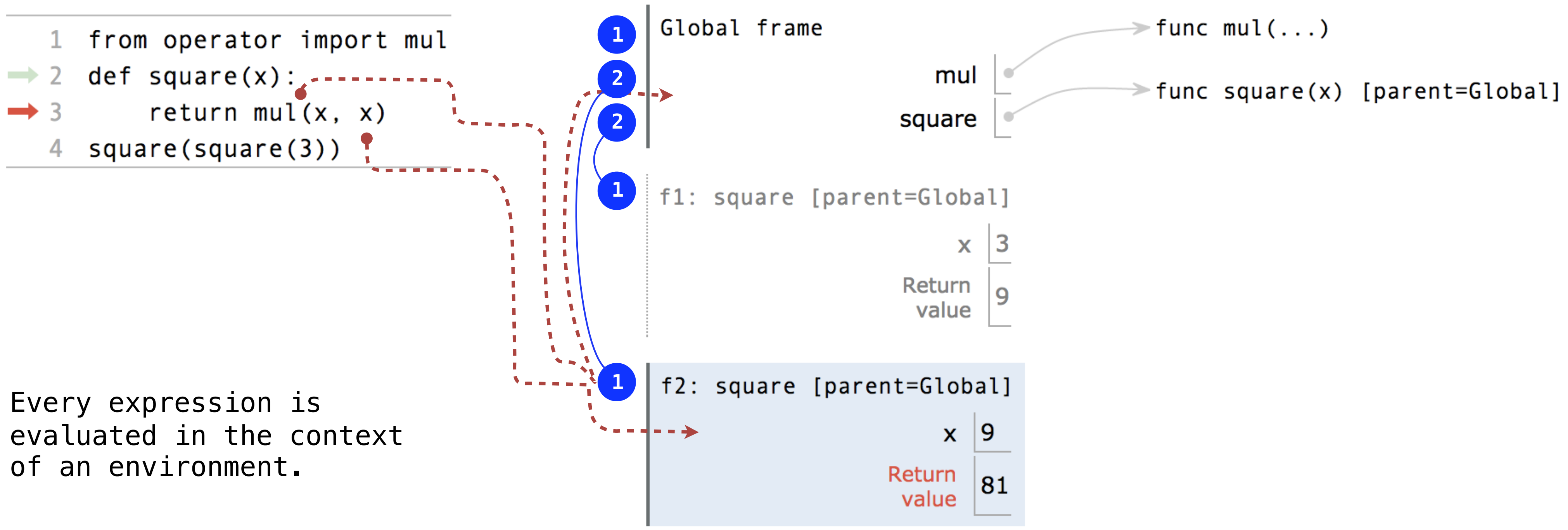


An environment is a sequence of frames.

- The global frame alone
- A local, then the global frame

Interactive Diagram

Names Have No Meaning Without Environments



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

Global frame
mul
square
func mul(...)
func square(x) [parent=Global]
f1: square [parent=Global]
x
3
Return value
9
f2: square [parent=Global]
x
9
Return value
81
1
2
2
1
1

Every expression is evaluated in the context of an environment.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

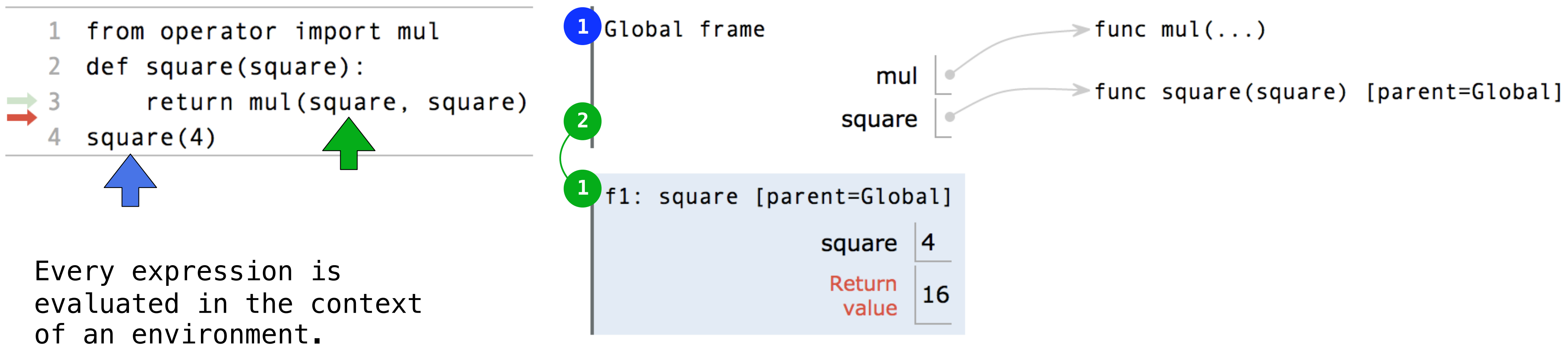
An environment is a sequence of frames.

- The global frame alone
- A local, then the global frame

Interactive Diagram

Names Have Different Meanings in Different Environments

A call expression and the body of the function being called
are evaluated in different environments



```
from operator import mul
def square(square):
    return mul(square, square)
square(4)
```

Global frame
mul
square
func mul(...)
func square(square) [parent=Global]
f1: square [parent=Global]
square
4
Return value
16
1
2
1

Every expression is evaluated in the context of an environment.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Interactive Diagram

Environments for Higher-Order Functions

Environments Enable Higher-Order Functions

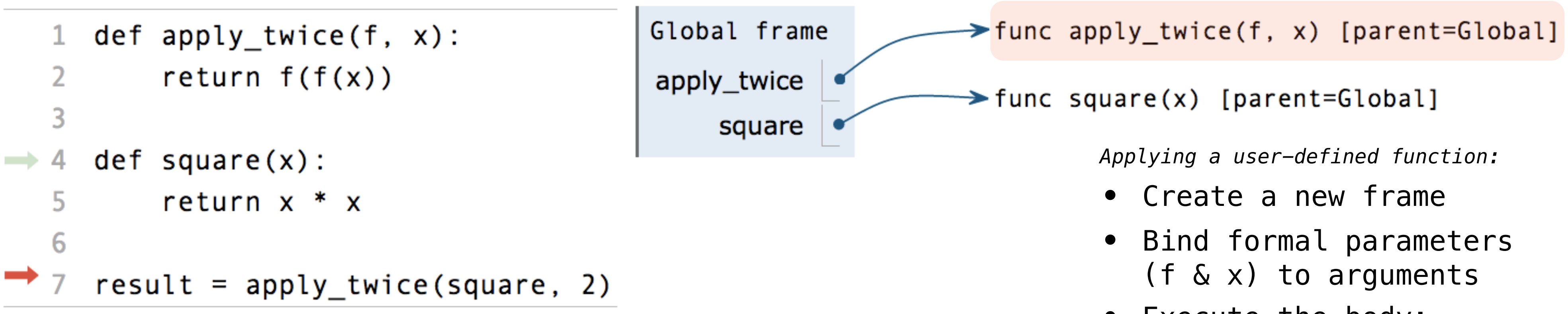
Functions are first-class: Functions are values in our programming language

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

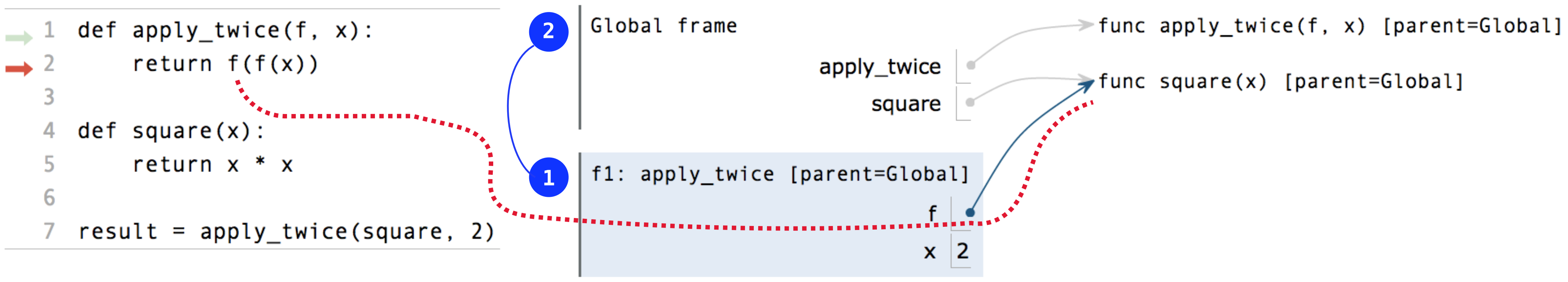


```
def apply_twice(f, x):
    return f(f(x))

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

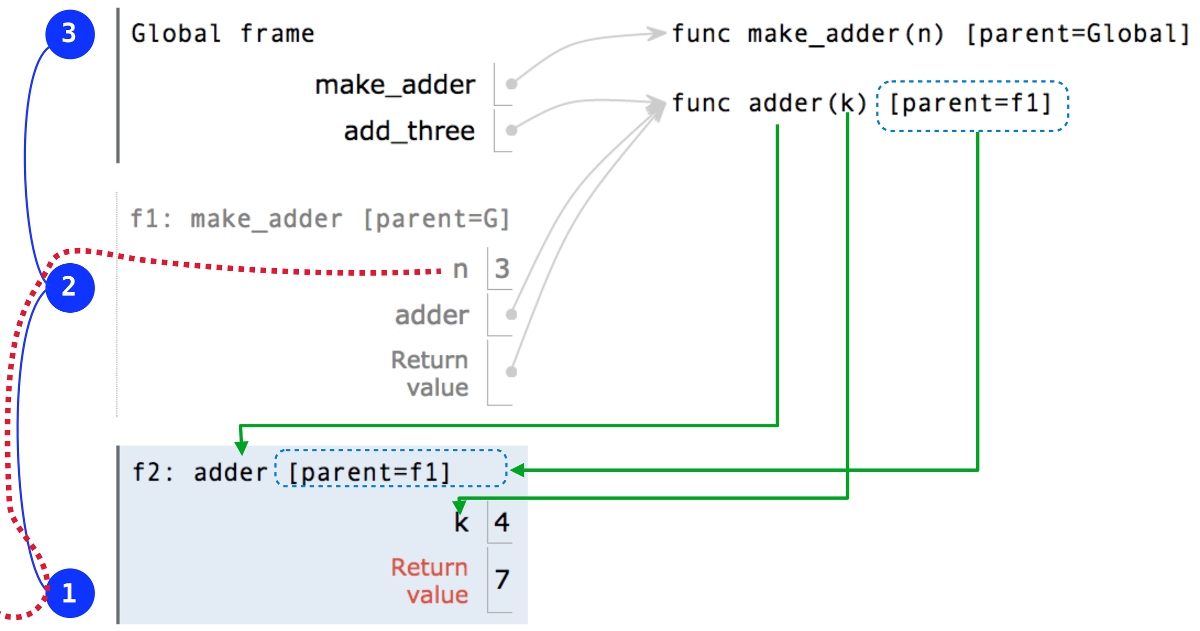
Environments for Nested Definitions

(Demo)

Environment Diagrams for Nested Def Statements

Nested def

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

add_three = make_adder(3)
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

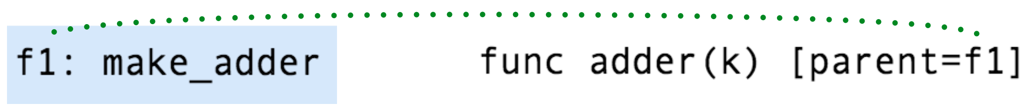
http://pythontutor.com/composingprograms.html#code=def%20make_adder%28n%29%3A%0A%20%20%20%20def%20adder%28k%29%3A%0A%20%20%20%20%20%20%20%20return%20k%20%2B%20n%0A%20%20%20%20return%20adder%0A%20%20%20%20%0Athree_more_than%20%3D%20make_adder%283%29%0Aresult%20%3D%20three_more_than%284%29&cumulative=false&curInstr=0&mode=display&origin=composingprograms.js&py=3&r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



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

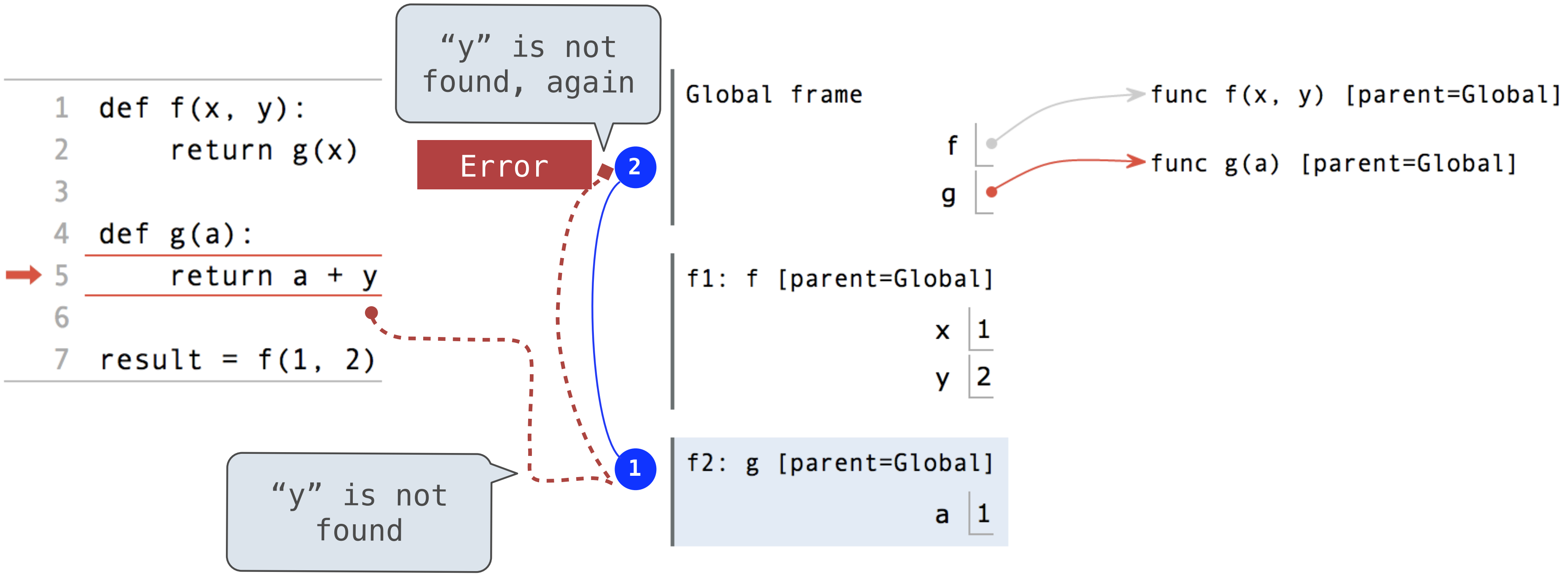
When a function is called:

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.

Lambda Expressions

(Demo)

Lambda Expressions

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Also an expression: evaluates to a function

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

Important: No "return" keyword!

A function

with formal parameter x

that returns the value of "x * x"

Must be a single expression

```
>>> square(4)
16
```

Lambda expressions are not common in Python, but important in general

Lambda expressions in Python cannot contain statements at all!

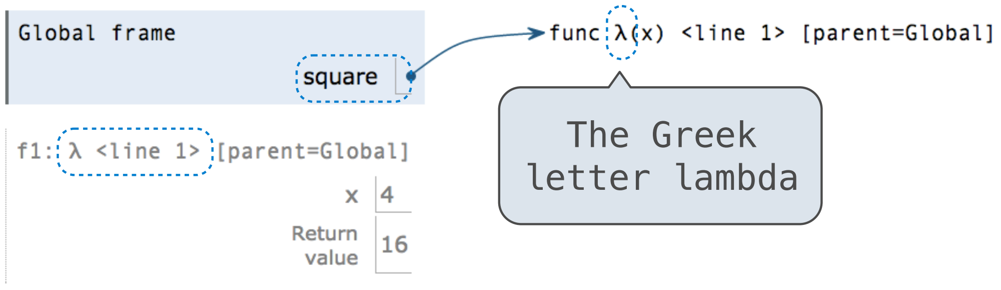
Lambda Expressions Versus Def Statements

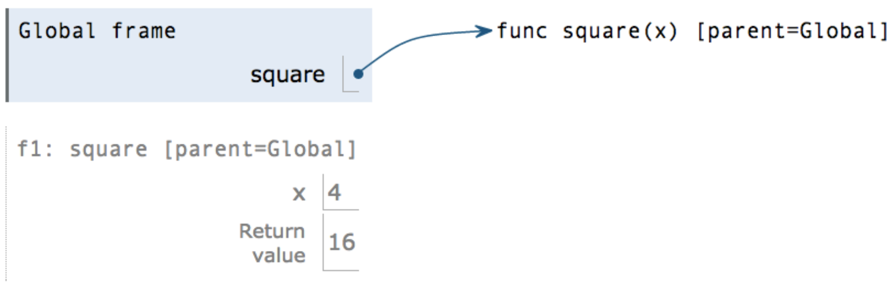
```
square = lambda x: x * x
```

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

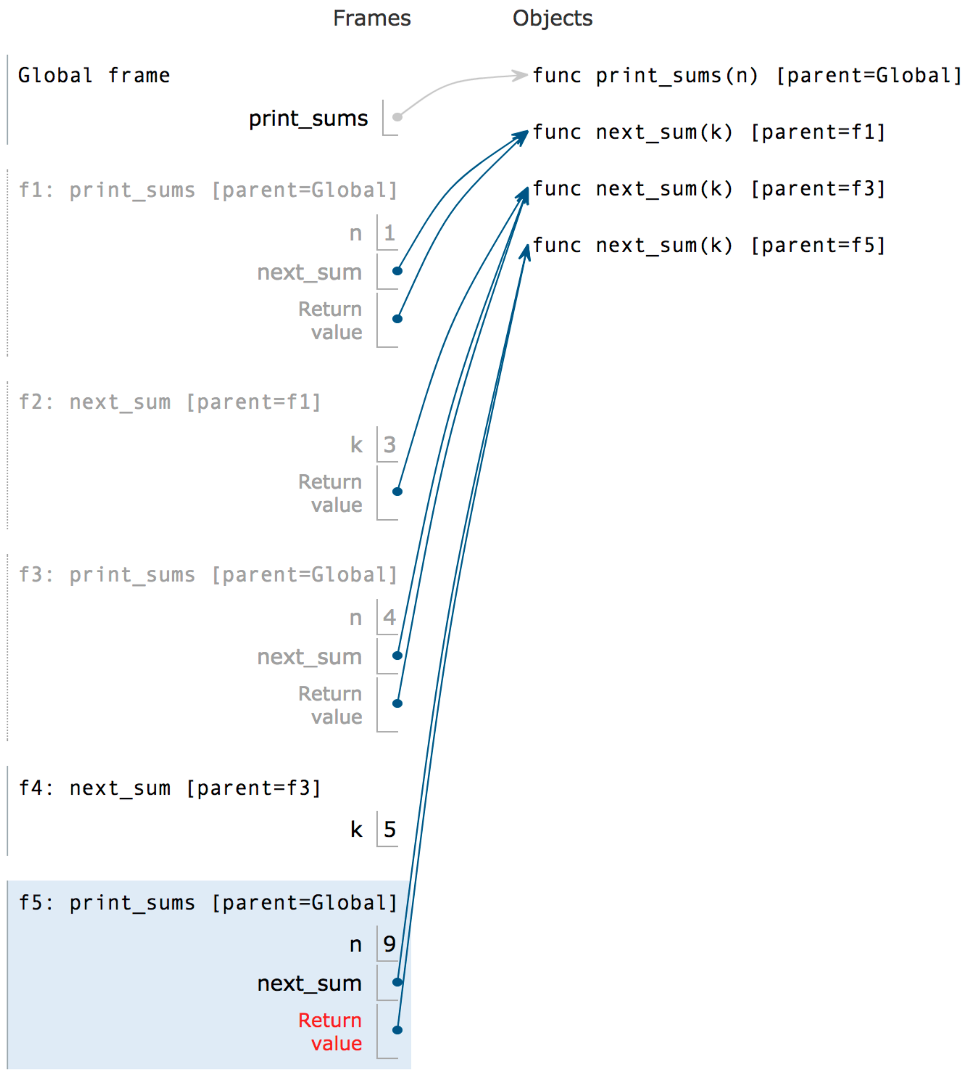


Self-Reference

`(Demo)`

Returning a Function Using Its Own Name

```python
def print_sums(n):
    print(n)
    def next_sum(k):
        return print_sums(n+k)
    return next_sum

print_sums(1)(3)(5)
```


Frames
Objects
Global frame
print_sums
func print_sums(n) [parent=Global]
func next_sum(k) [parent=f1]
func next_sum(k) [parent=f3]
func next_sum(k) [parent=f5]
f1: print_sums [parent=Global]
n 1
next_sum
Return value
f2: next_sum [parent=f1]
k 3
Return value
f3: print_sums [parent=Global]
n 4
next_sum
Return value
f4: next_sum [parent=f3]
k 5
f5: print_sums [parent=Global]
n 9
next_sum
Return value

Review

What Would Python Display?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that takes any argument and returns a function that returns that arg

```
def delay(arg):
    print('delayed')
    def g():
        return arg
    return g
```

Names in nested def statements can refer to their enclosing scope

This expression	Evaluates to	Interactive Output
5	5	5
print(5)	None	5
print($\underbrace{\text{print(5)}}_{\text{None}}$)	None	5 None
delay(delay)()(6)()	6	delayed delayed 6
print(delay(print)()(4))	None	delayed 4 None

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

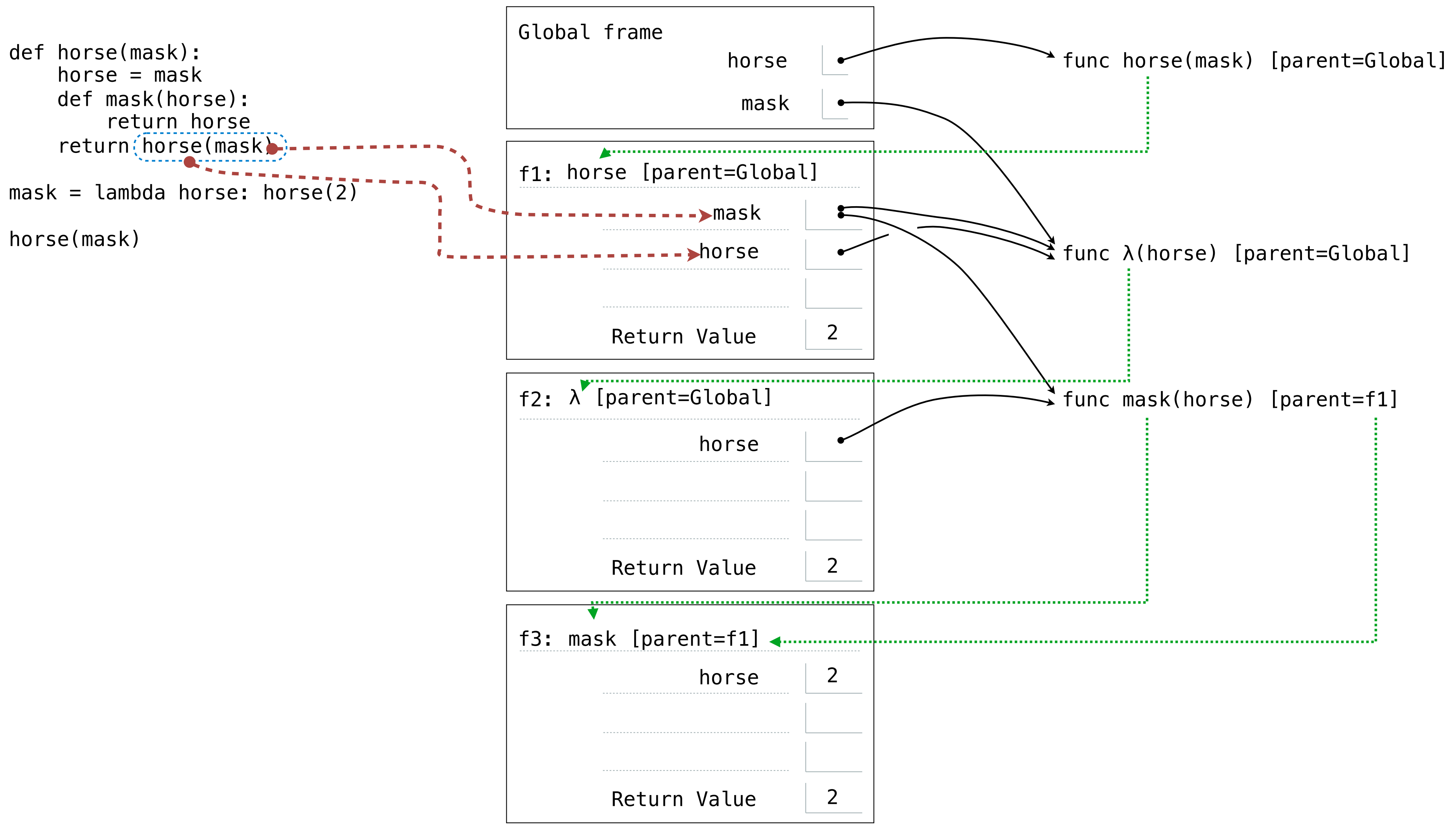
```
from operator import add, mul
def square(x):
    return mul(x, x)
```

A function that always returns the identity function

```
def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression	Evaluates to	Interactive Output
add(pirate(3)(square)(4), 1) *func square(x)* *16*	17	Matey 17
pirate(pirate(pirate))(5)(7) *Identity function* *5*	Error	Matey Matey Error

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.



```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```