61A Lecture 5

Wednesday, September 10

Announcements

Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm

Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - http://cs61a.org/hw/released/quiz1.html

Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - http://cs61a.org/hw/released/quiz1.html
  - 3 points; graded for correctness

Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - http://cs61a.org/hw/released/quiz1.html
  - 3 points; graded for correctness
  - Submit in the same way that you submit homework assignments

Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - http://cs61a.org/hw/released/quiz1.html
  - 3 points; graded for correctness
  - Submit in the same way that you submit homework assignments
  - If you receive 0/3, you will need to talk to the course staff or be dropped

Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - http://cs61a.org/hw/released/quiz1.html
  - 3 points; graded for correctness
  - Submit in the same way that you submit homework assignments
  - If you receive 0/3, you will need to talk to the course staff or be dropped
  - Open computer & course materials, but no external resources such as classmates

Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - http://cs61a.org/hw/released/quiz1.html
  - 3 points; graded for correctness
  - Submit in the same way that you submit homework assignments
  - If you receive 0/3, you will need to talk to the course staff or be dropped
  - Open computer & course materials, but no external resources such as classmates
  - Practice quiz from Fall 2013: http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html

Announcements

- Take–home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - http://cs61a.org/hw/released/quiz1.html
  - 3 points; graded for correctness
  - Submit in the same way that you submit homework assignments
  - If you receive 0/3, you will need to talk to the course staff or be dropped
  - Open computer & course materials, but no external resources such as classmates
  - Practice quiz from Fall 2013: http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html
- "Practical Programming Skills" DeCal starts Thursday 9/11, 6:30pm to 8pm in 306 Soda

Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - http://cs61a.org/hw/released/quiz1.html
  - 3 points; graded for correctness
  - Submit in the same way that you submit homework assignments
  - If you receive 0/3, you will need to talk to the course staff or be dropped
  - Open computer & course materials, but no external resources such as classmates
  - Practice quiz from Fall 2013: http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html
- "Practical Programming Skills" DeCal starts Thursday 9/11, 6:30pm to 8pm in 306 Soda
  - http://42.cs61a.org, run by Sumukh Sridhara (TA)

Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - http://cs61a.org/hw/released/quiz1.html
  - 3 points; graded for correctness
  - Submit in the same way that you submit homework assignments
  - If you receive 0/3, you will need to talk to the course staff or be dropped
  - Open computer & course materials, but no external resources such as classmates
  - Practice quiz from Fall 2013: http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html
- "Practical Programming Skills" DeCal starts Thursday 9/11, 6:30pm to 8pm in 306 Soda
  - http://42.cs61a.org, run by Sumukh Sridhara (TA)
- Guerrilla Section 1 on Higher-order functions: Saturday 9/13, 12:30pm to 3pm in 306 Soda

Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - http://cs61a.org/hw/released/quiz1.html
  - 3 points; graded for correctness
  - Submit in the same way that you submit homework assignments
  - If you receive 0/3, you will need to talk to the course staff or be dropped
  - Open computer & course materials, but no external resources such as classmates
  - Practice quiz from Fall 2013: http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html
- "Practical Programming Skills" DeCal starts Thursday 9/11, 6:30pm to 8pm in 306 Soda
  - http://42.cs61a.org, run by Sumukh Sridhara (TA)
- Guerrilla Section 1 on Higher-order functions: Saturday 9/13, 12:30pm to 3pm in 306 Soda
- Homework 2 (which is small) due Monday 9/15 at 11:59pm.

Announcements

- Take-home quiz released Wednesday 9/10 at 3pm, due Thursday 9/11 at 11:59pm
  - http://cs61a.org/hw/released/quiz1.html
  - 3 points; graded for correctness
  - Submit in the same way that you submit homework assignments
  - If you receive 0/3, you will need to talk to the course staff or be dropped
  - Open computer & course materials, but no external resources such as classmates
  - Practice quiz from Fall 2013: http://inst.eecs.berkeley.edu/~cs61a/fa13/hw/quiz1.html
- "Practical Programming Skills" DeCal starts Thursday 9/11, 6:30pm to 8pm in 306 Soda
  - http://42.cs61a.org, run by Sumukh Sridhara (TA)
- Guerrilla Section 1 on Higher-order functions: Saturday 9/13, 12:30pm to 3pm in 306 Soda
- Homework 2 (which is small) due Monday 9/15 at 11:59pm.
- Project 1 (which is BIG) due Wednesday 9/17 at 11:59pm.

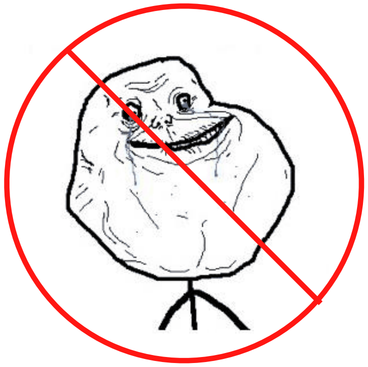
Office Hours: You Should Go!

Office Hours: You Should Go!

You are not alone!

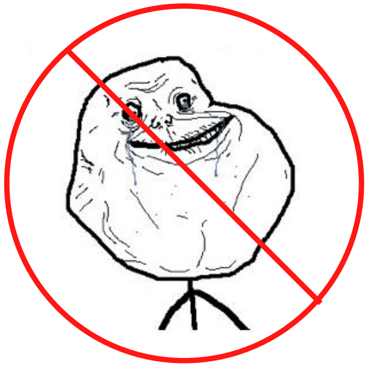
Office Hours: You Should Go!

You are not alone!



Office Hours: You Should Go!

You are not alone!



http://cs61a.org/staff.html

Environments for Higher-Order Functions

Environments Enable Higher-Order Functions

Environments Enable Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

Environments Enable Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

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

Environments Enable Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

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

Higher-order functions:

Environments Enable Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

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

Higher-order functions:

- Express general methods of computation

Environments Enable Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

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

Higher-order functions:

- Express general methods of computation
- Remove repetition from programs

Environments Enable Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

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

Higher-order functions:

- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions

Environments Enable Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

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

Higher-order functions:

- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions

Environment diagrams describe how higher-order functions work!

Environments Enable Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

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

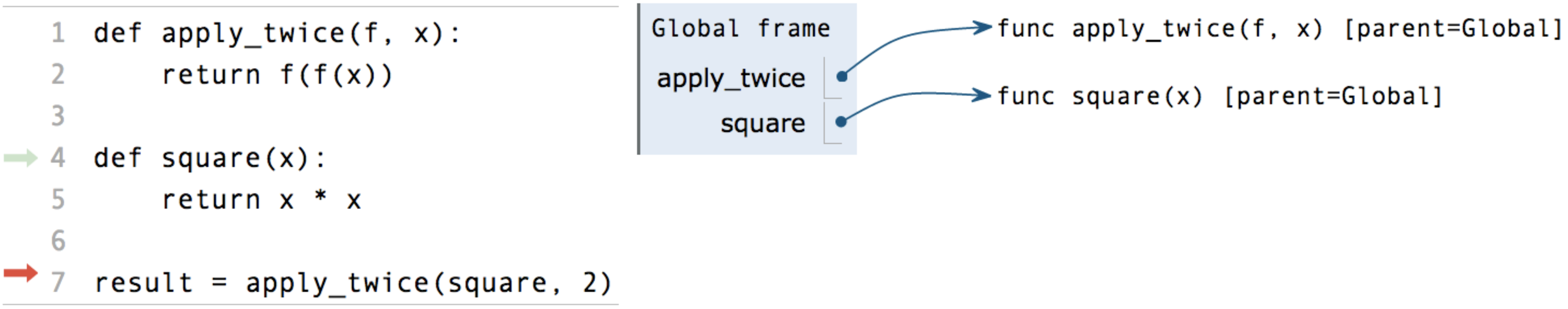
Higher-order functions:

- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions

Environment diagrams describe how higher-order functions work!

(Demo)

Names can be Bound to Functional Arguments

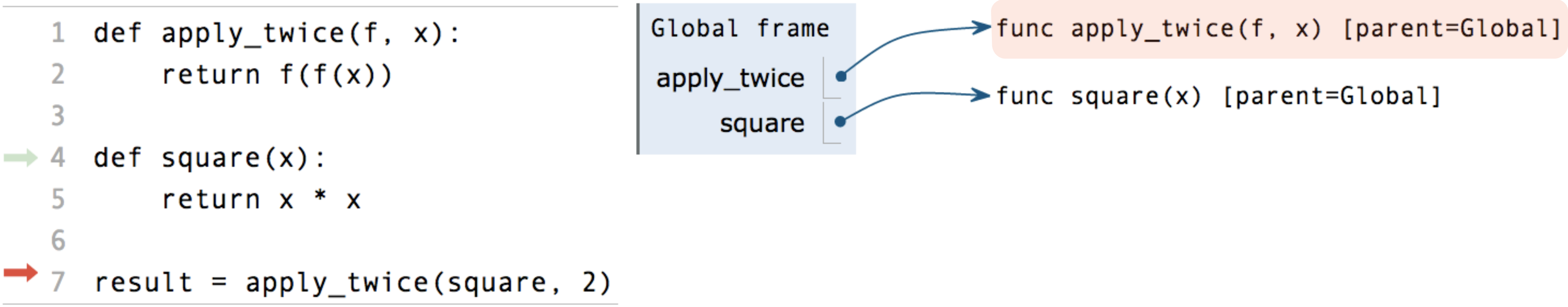


```
def apply_twice(f, x):
    return f(f(x))

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]

Interactive Diagram

Names can be Bound to Functional Arguments

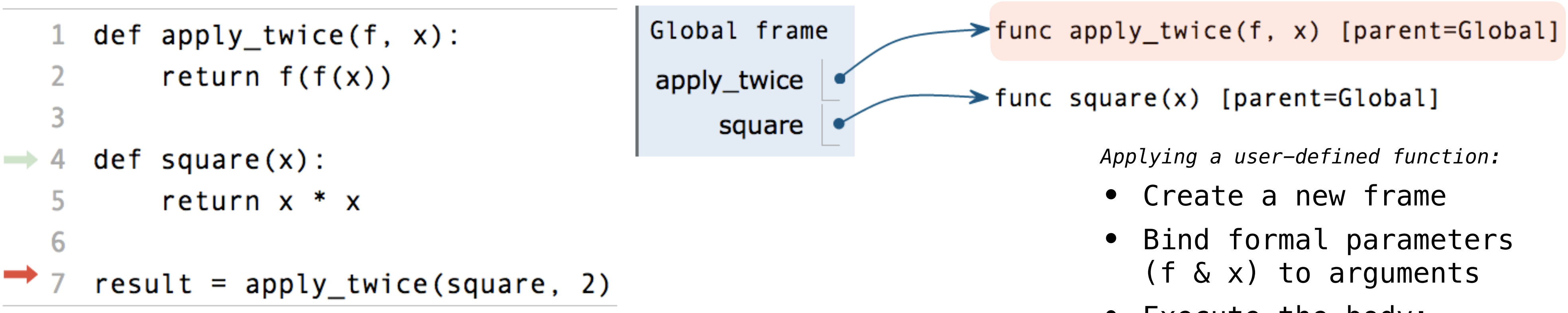


```
def apply_twice(f, x):
    return f(f(x))

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]

Interactive Diagram

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

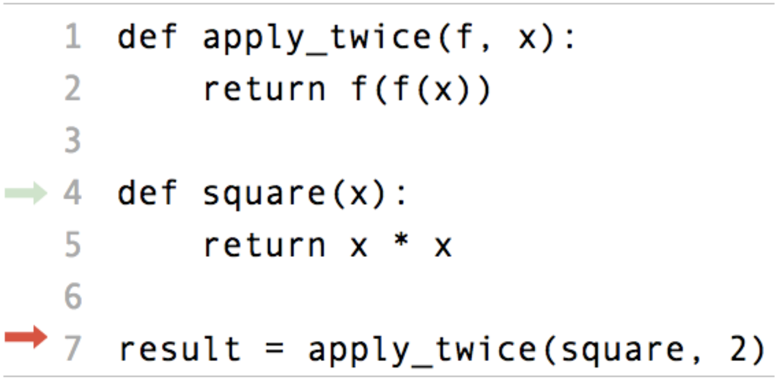
square → func square(x) [parent=Global]

Applying a user-defined function:

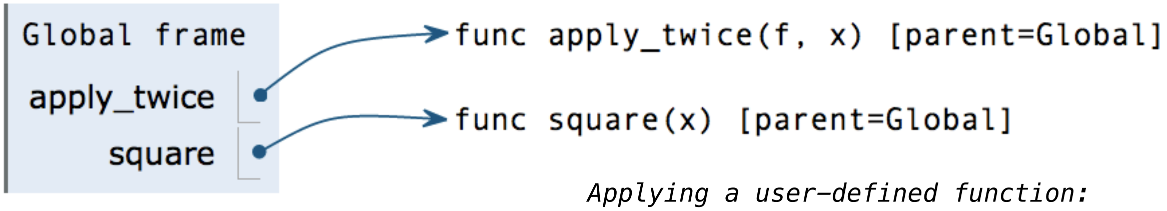
- Create a new frame
- Bind formal parameters (f & x) to arguments
- Execute the body: return f(f(x))

Interactive Diagram

Names can be Bound to Functional Arguments

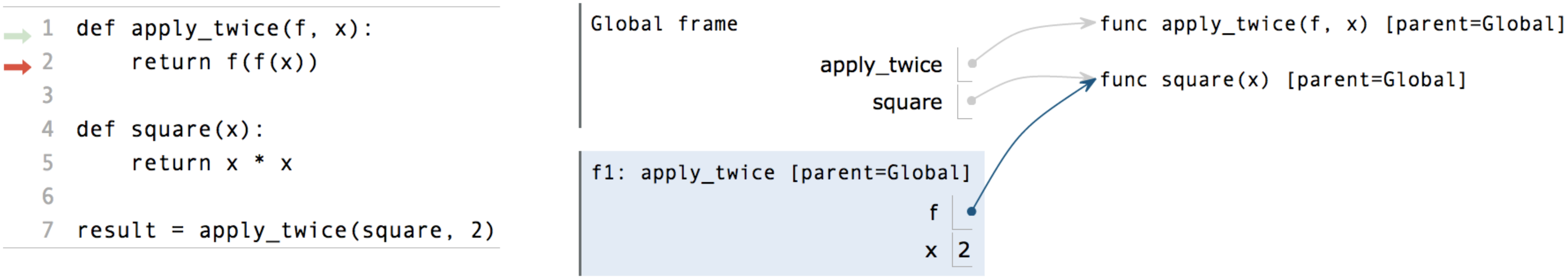


```
def apply_twice(f, x):
    return f(f(x))

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))

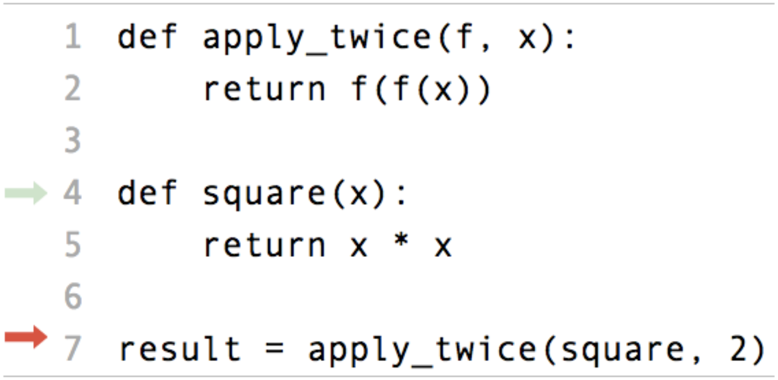


```
def apply_twice(f, x):
    return f(f(x))

def square(x):
    return x * x

result = apply_twice(square, 2)
```

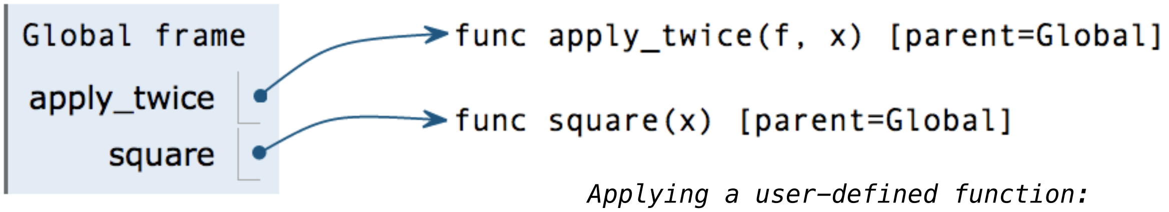
Global frame
apply_twice
square
func apply_twice(f, x) [parent=Global]
func square(x) [parent=Global]
f1: apply_twice [parent=Global]
f
x
2

Interactive Diagram

Names can be Bound to Functional Arguments

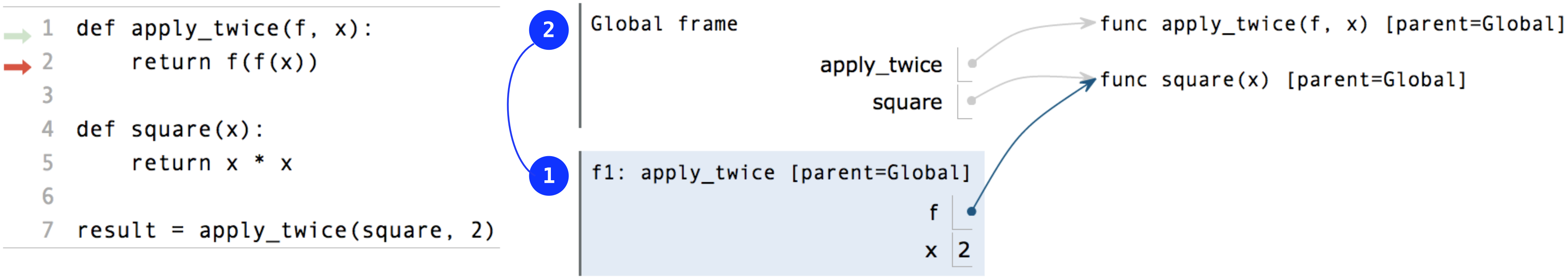


```python
def apply_twice(f, x):
    return f(f(x))

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))

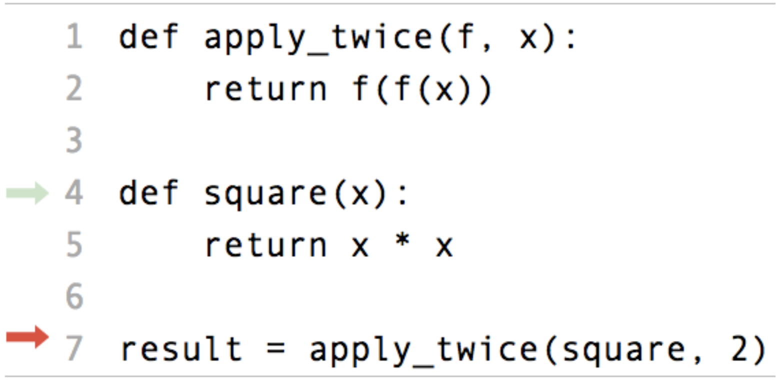


```python
def apply_twice(f, x):
    return f(f(x))

def square(x):
    return x * x

result = apply_twice(square, 2)
```

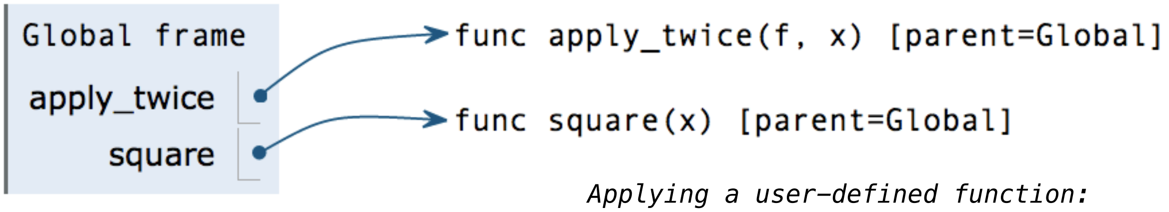
2
1
Global frame
apply_twice
square
func apply_twice(f, x) [parent=Global]
func square(x) [parent=Global]
f1: apply_twice [parent=Global]
f
x
2

Interactive Diagram

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)
```



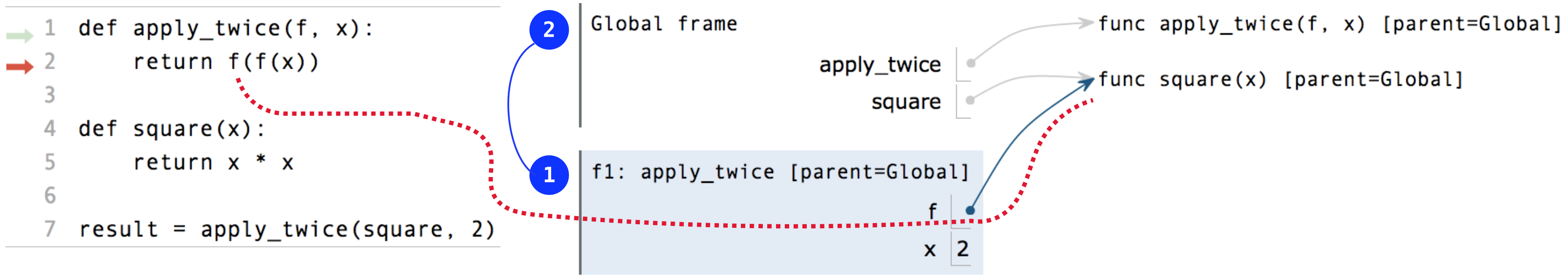
Global frame

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

square → 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)
```

Global frame

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

square → func square(x) [parent=Global]

f1: apply_twice [parent=Global]

f → func square(x) [parent=Global]

x 2

Interactive Diagram

Discussion Question

What is the value of the final expression below? (Demo)

Interactive Diagram

Discussion Question

What is the value of the final expression below? (Demo)

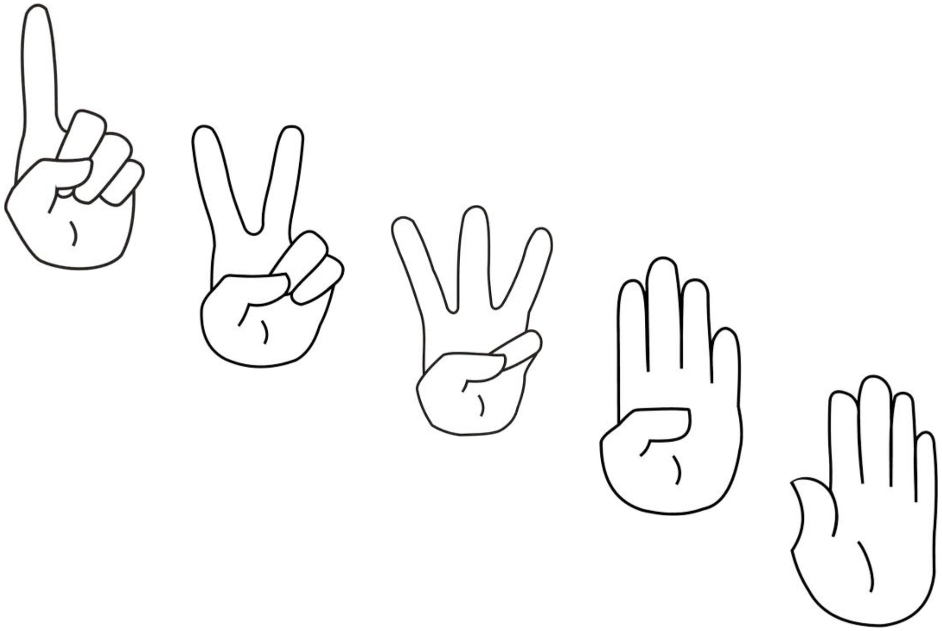
```
def repeat(f, x):
    while f(x) != x:
        x = f(x)
    return x

def g(y):
    return (y + 5) // 3

result = repeat(g, 5)
```

Interactive Diagram

Discussion Question

What is the value of the final expression below? (Demo)

```
def repeat(f, x):
    while f(x) != x:
        x = f(x)
    return x

def g(y):
    return (y + 5) // 3

result = repeat(g, 5)
```

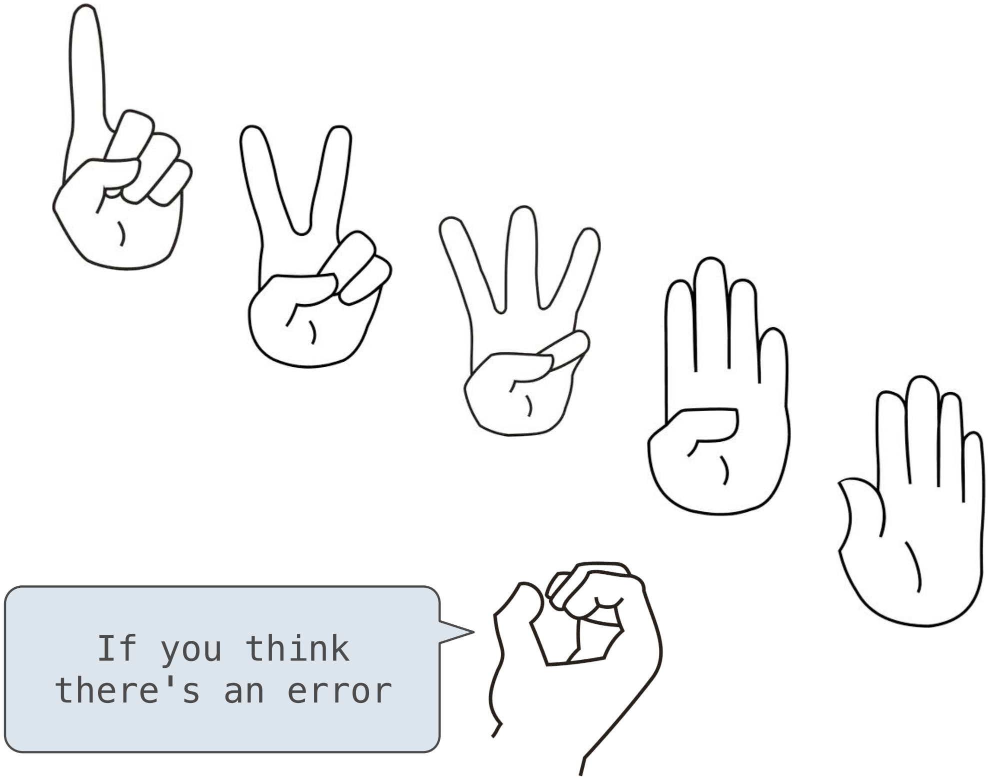


Interactive Diagram

Discussion Question

What is the value of the final expression below? (Demo)

```
def repeat(f, x):
    while f(x) != x:
        x = f(x)
    return x

def g(y):
    return (y + 5) // 3

result = repeat(g, 5)
```

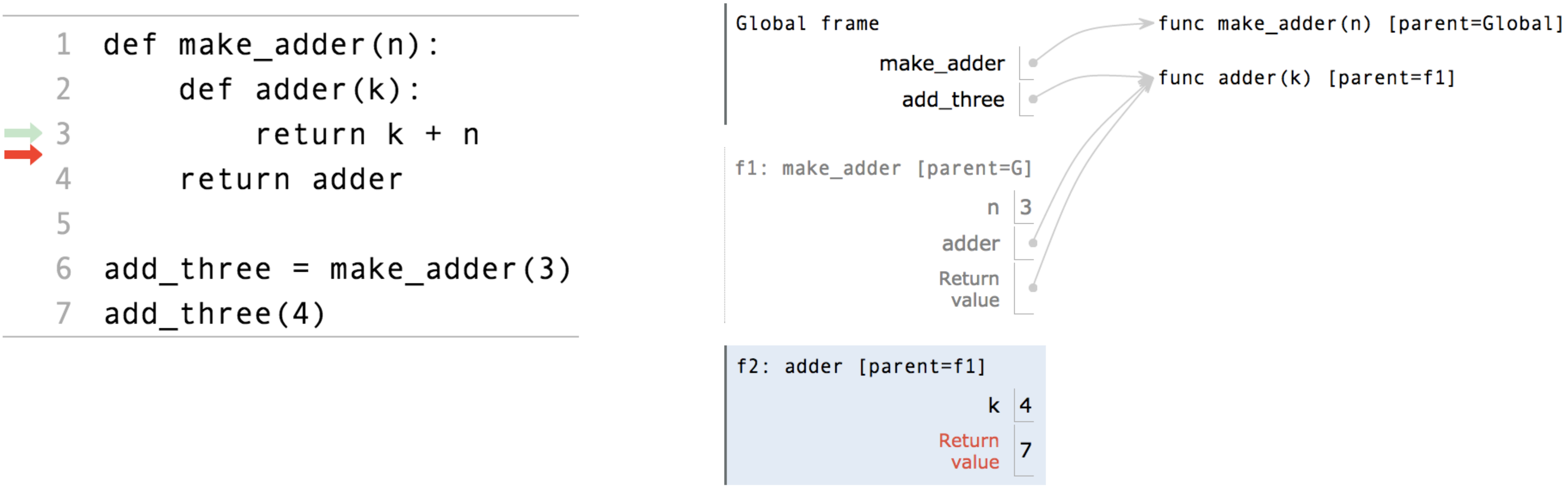


Interactive Diagram

Environments for Nested Definitions

(Demo)

Environment Diagrams for Nested Def Statements



```
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

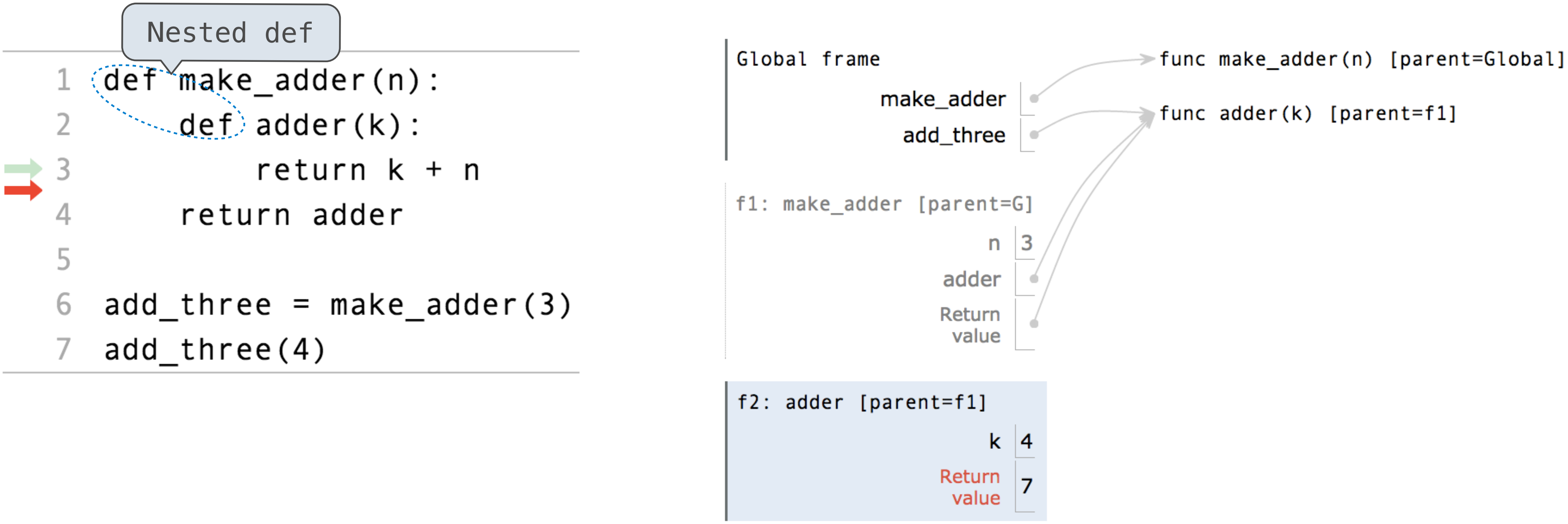
f2: adder [parent=f1]

k 4

Return value 7

Interactive Diagram

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	→ func make_adder(n) [parent=Global]
add_three	→ func adder(k) [parent=f1]

f1: make_adder [parent=G]

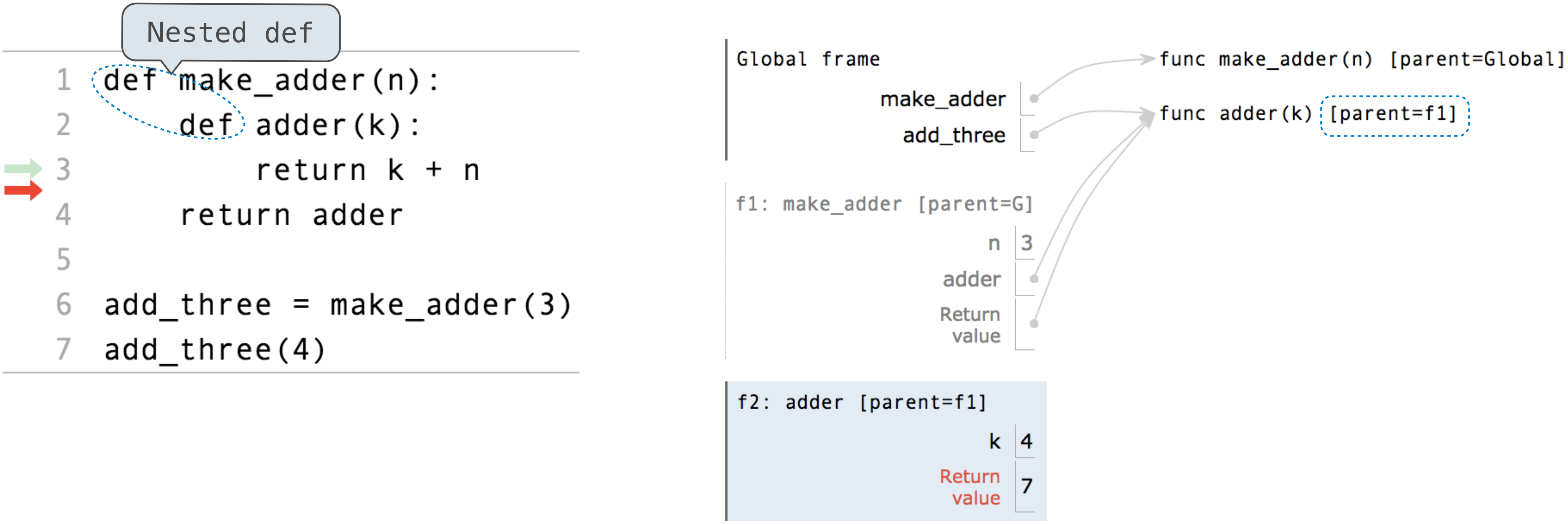
n	3
adder	→ func adder(k) [parent=f1]
Return value	→ func adder(k) [parent=f1]

f2: adder [parent=f1]

k	4
Return value	7

Interactive Diagram

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

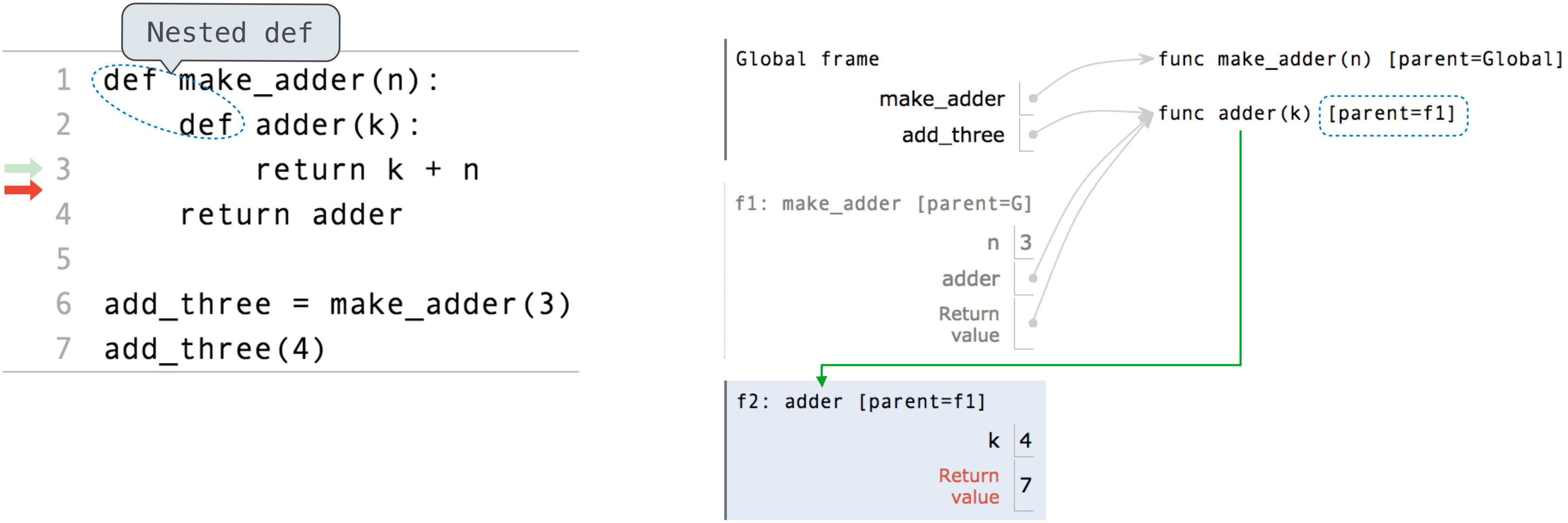
f2: adder [parent=f1]

k 4

Return value 7

Interactive Diagram

Environment Diagrams for Nested Def Statements



```
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

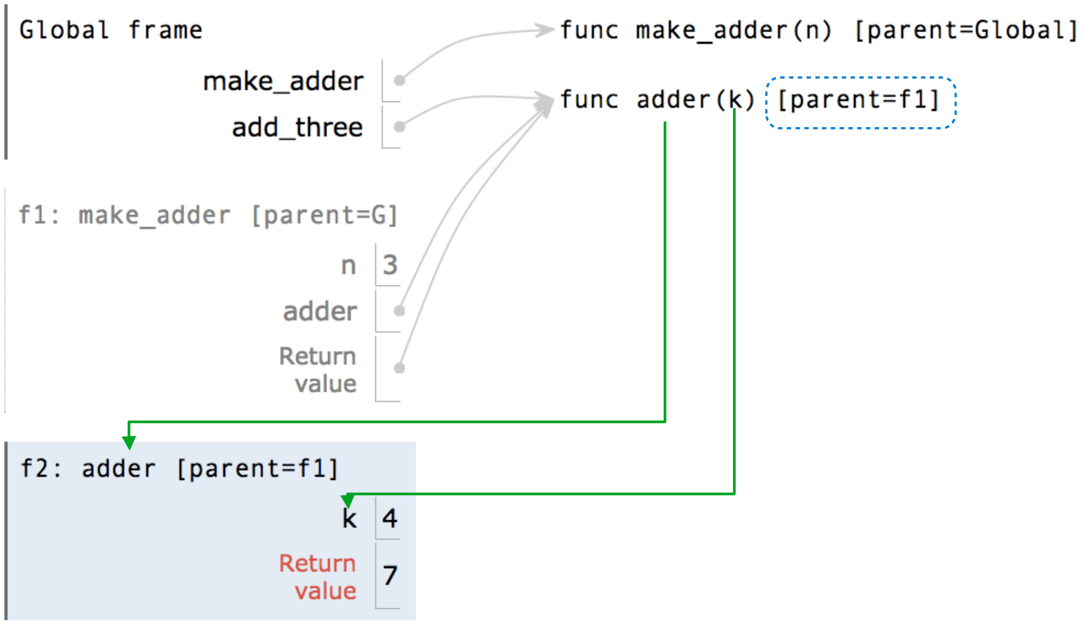
Return value 7

Interactive Diagram

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)
```

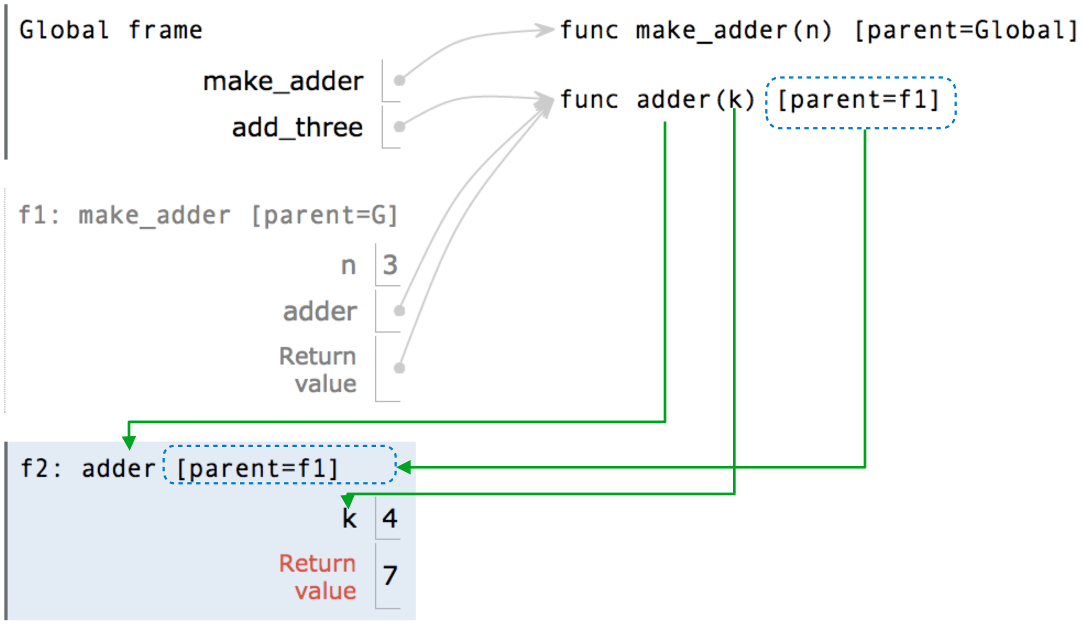


Interactive Diagram

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

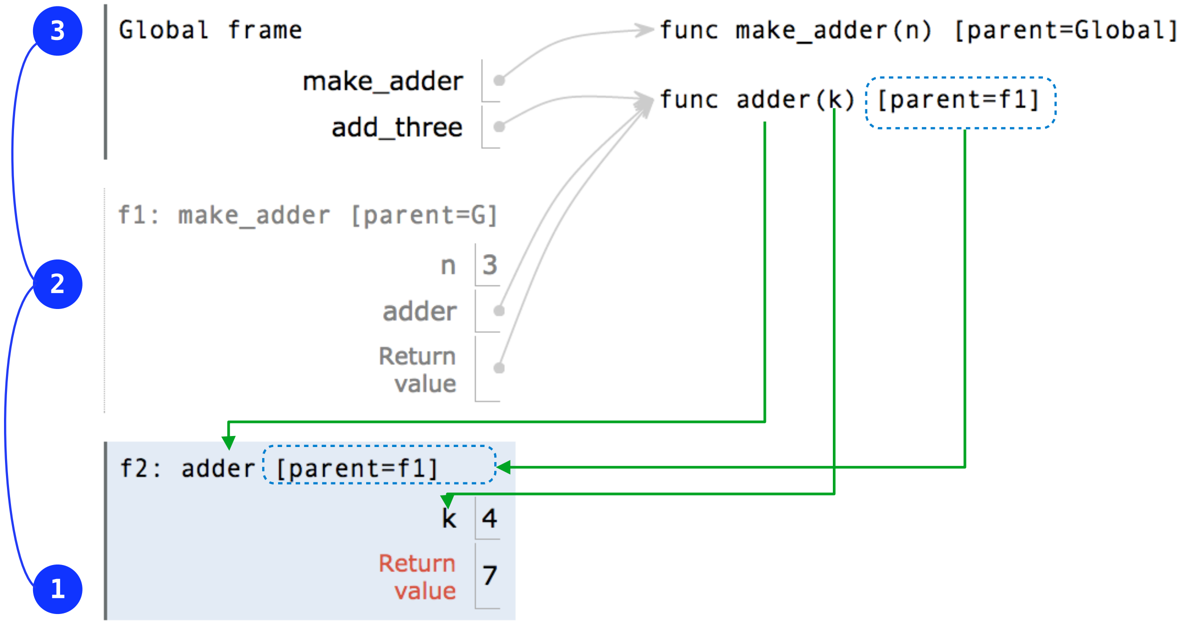
Return value 7

Interactive Diagram

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)
```



3

Global frame

make_adder

add_three

func make_adder(n) [parent=Global]

func adder(k) [parent=f1]

2

f1: make_adder [parent=G]

n 3

adder

Return value

1

f2: adder [parent=f1]

k 4

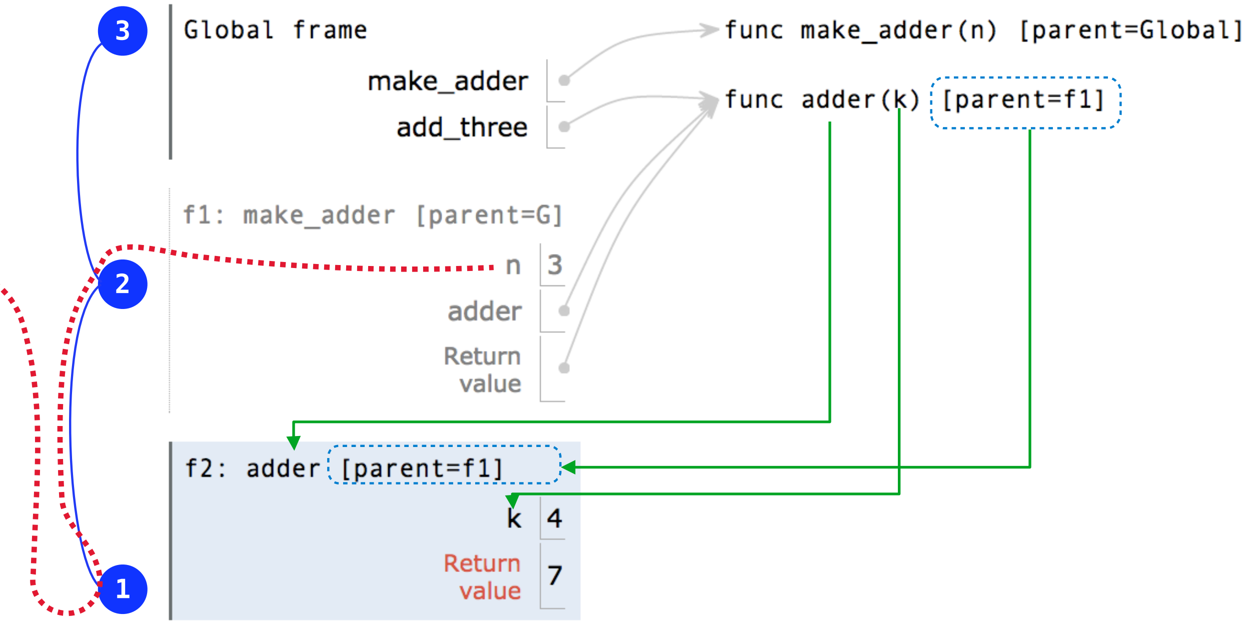
Return value 7

Interactive Diagram

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]

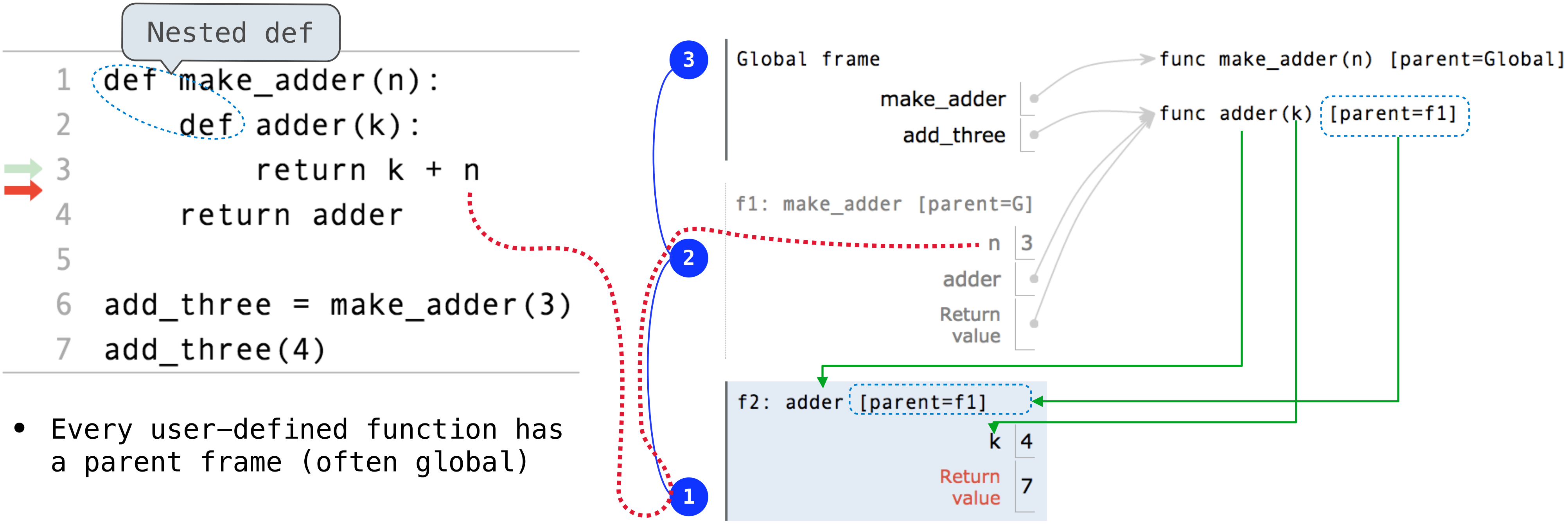
n	3
adder	
Return value	

f2: adder [parent=f1]

k	4
Return value	7

Interactive Diagram

Environment Diagrams for Nested Def Statements



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

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

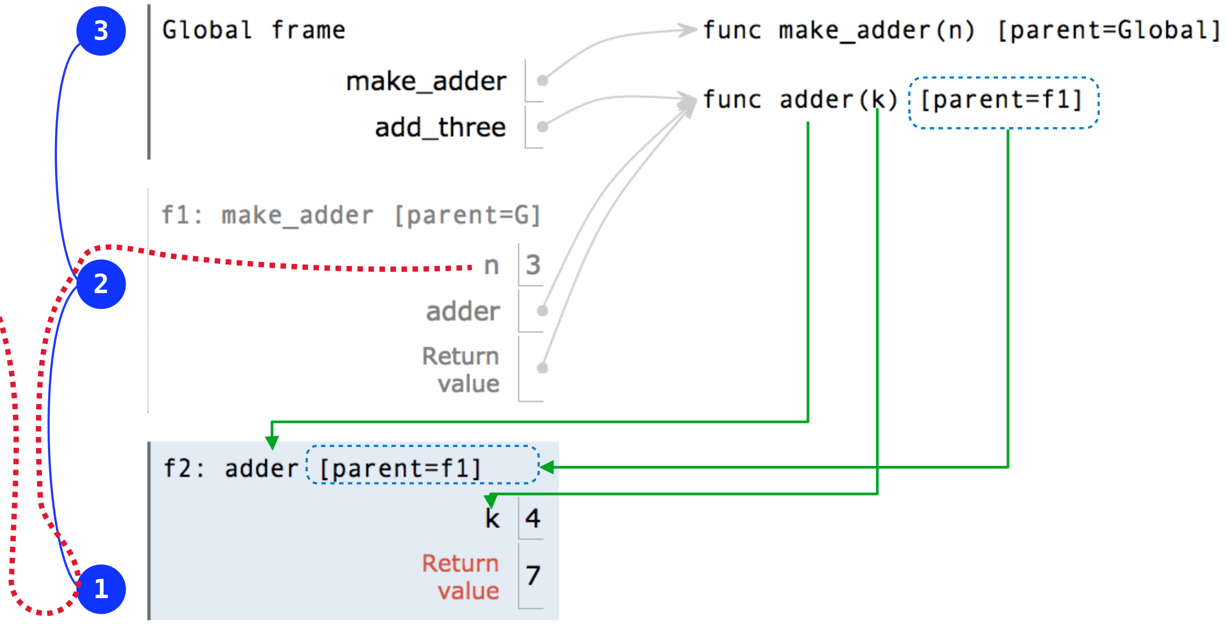
- Every user-defined function has a parent frame (often global)

Interactive Diagram

Environment Diagrams for Nested Def Statements

Nested def

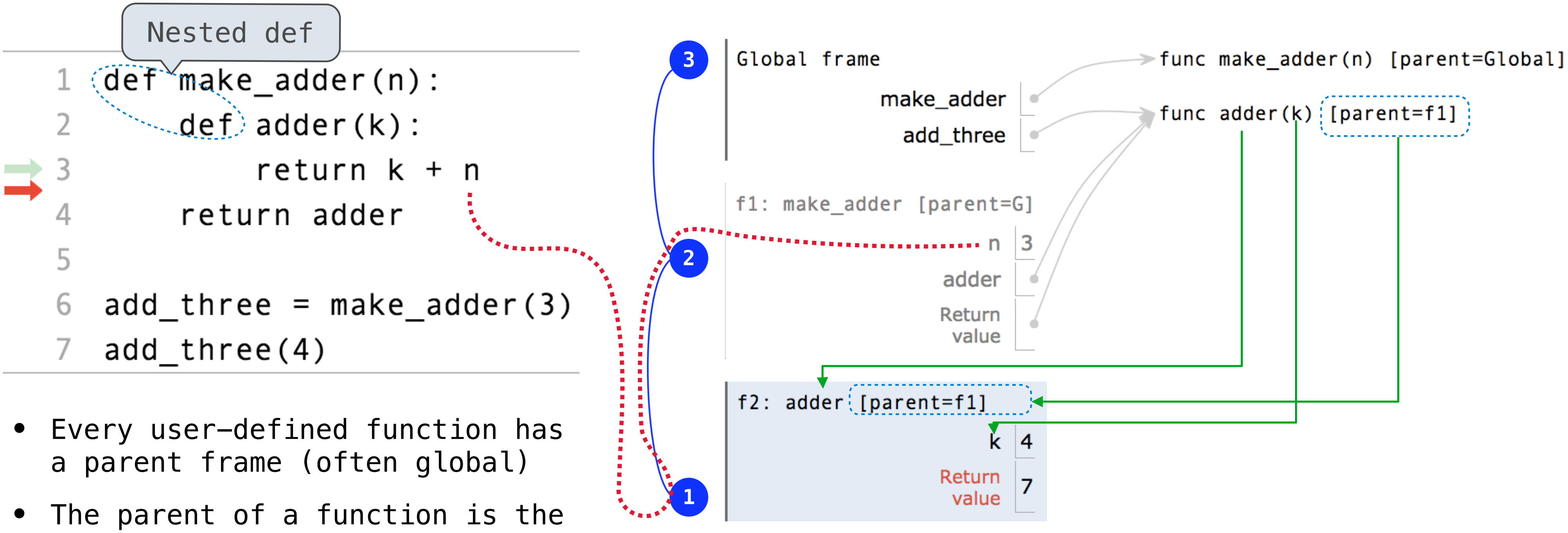
```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

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



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

Interactive Diagram

Environment Diagrams for Nested Def Statements

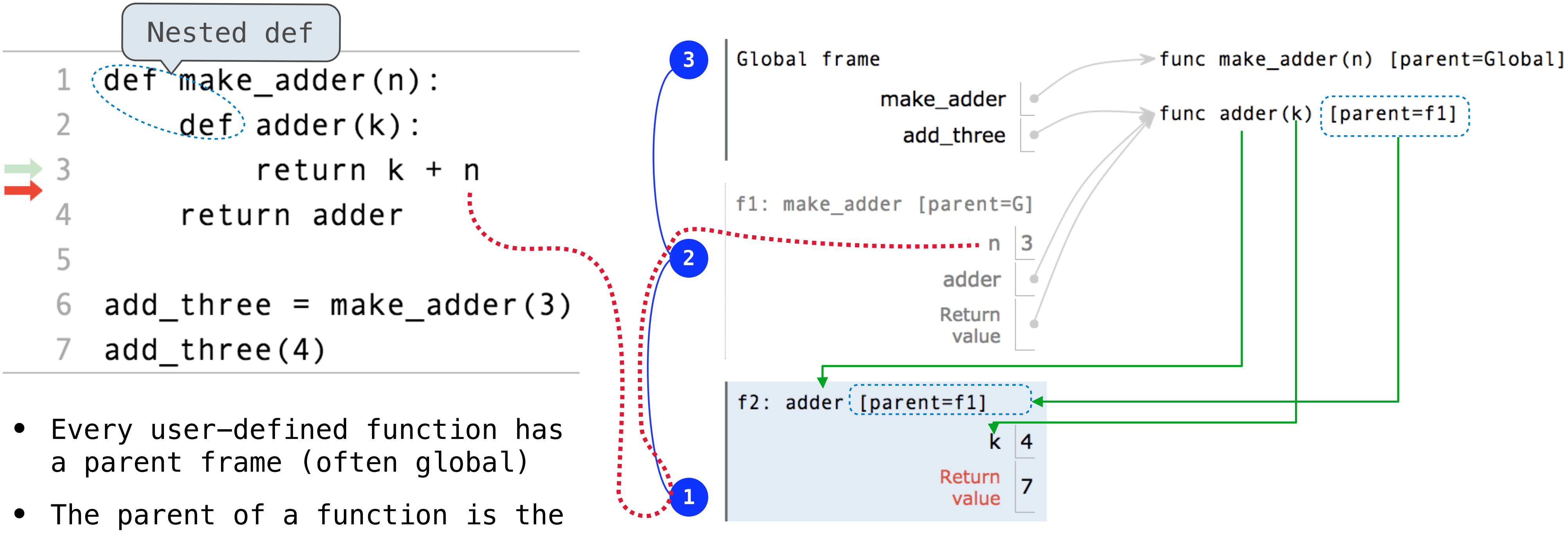


```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

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)

Interactive Diagram

Environment Diagrams for Nested Def Statements



```
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

Interactive Diagram

How to Draw an Environment Diagram

How to Draw an Environment Diagram

When a function is defined:

How to Draw an Environment Diagram

When a function is defined:

Create a function value: func <name>(<formal parameters>) [parent=<label>]

How to Draw an Environment Diagram

When a function is defined:

Create a function value: `func <name>(<formal parameters>) [parent=<label>]`

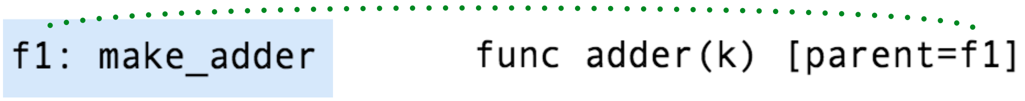
Its parent is the current frame.

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

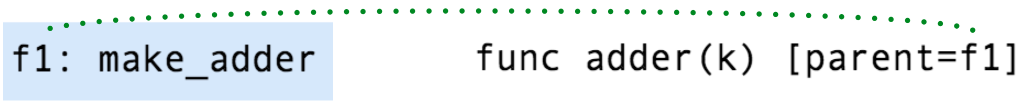
func adder(k) [parent=f1]

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]

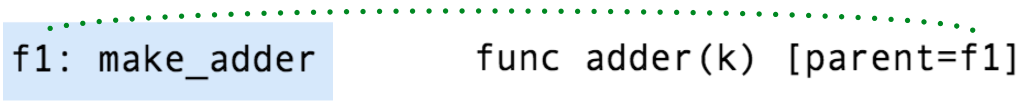
Bind <name> to the function value in the current frame

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

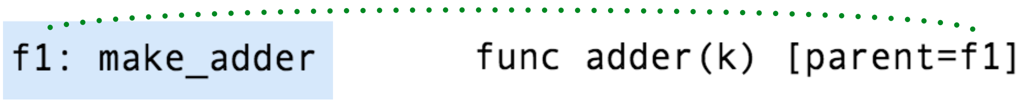
When a function is called:

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:

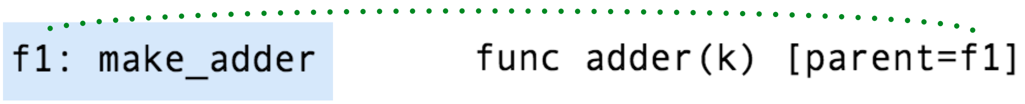
1. Add a local frame, titled with the <name> of the function being called.

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:

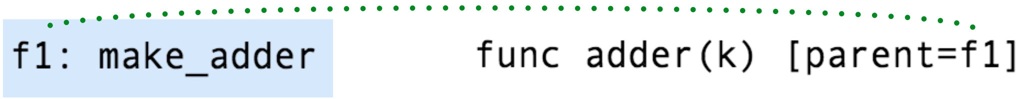
1. Add a local frame, titled with the <name> of the function being called.
2. Copy the parent of the function to the local frame: [parent=<label>]

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:

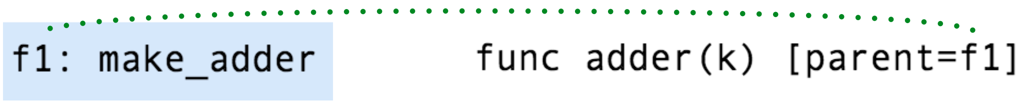
1. Add a local frame, titled with the <name> of the function being called.
2. Copy the parent of the function to the local frame: [parent=<label>]
3. Bind the <formal parameters> to the arguments in the local frame.

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

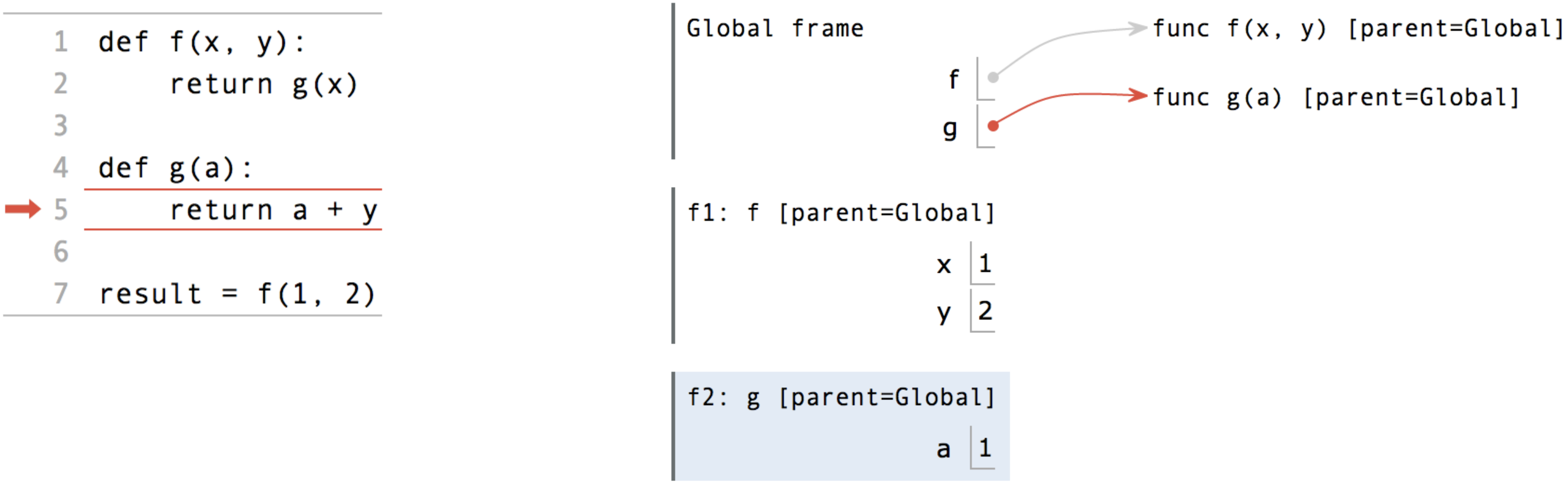
When a function is called:

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)

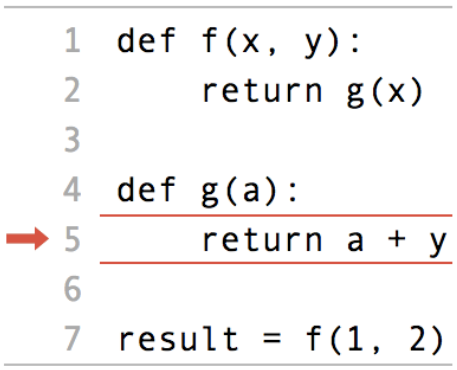
Local Names are not Visible to Other (Non-Nested) Functions



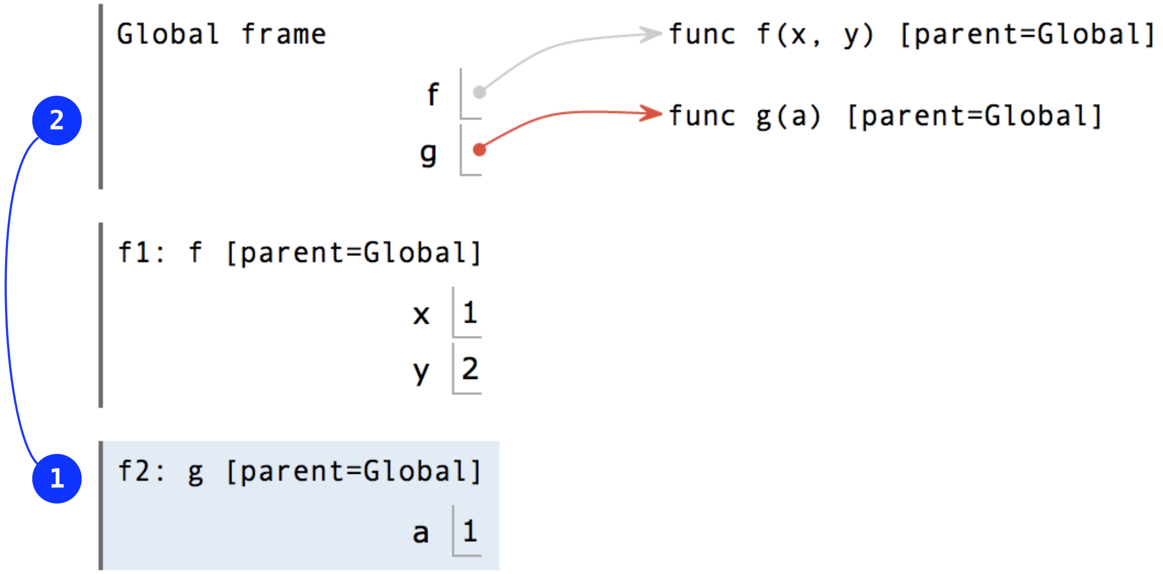
```
def f(x, y):
    return g(x)

def g(a):
    return a + y

result = f(1, 2)
```

Interactive Diagram

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

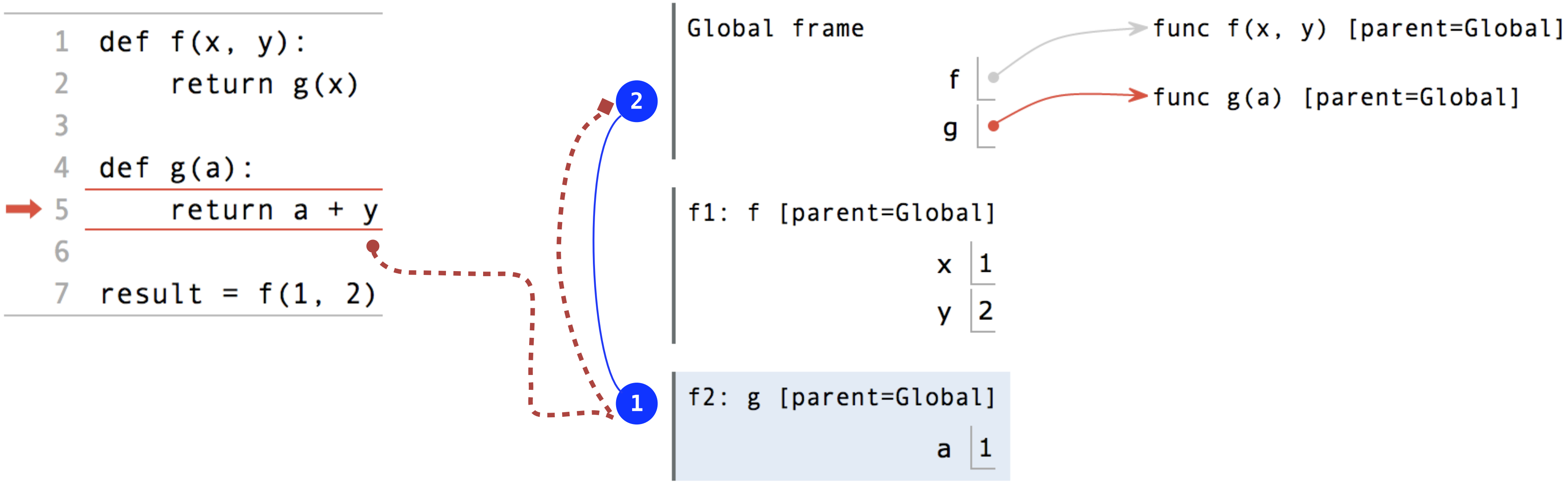


```
def f(x, y):
    return g(x)

def g(a):
    return a + y

result = f(1, 2)
```



Interactive Diagram

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)
```

Global frame

f → func f(x, y) [parent=Global]

g → func g(a) [parent=Global]

f1: f [parent=Global]

x 1

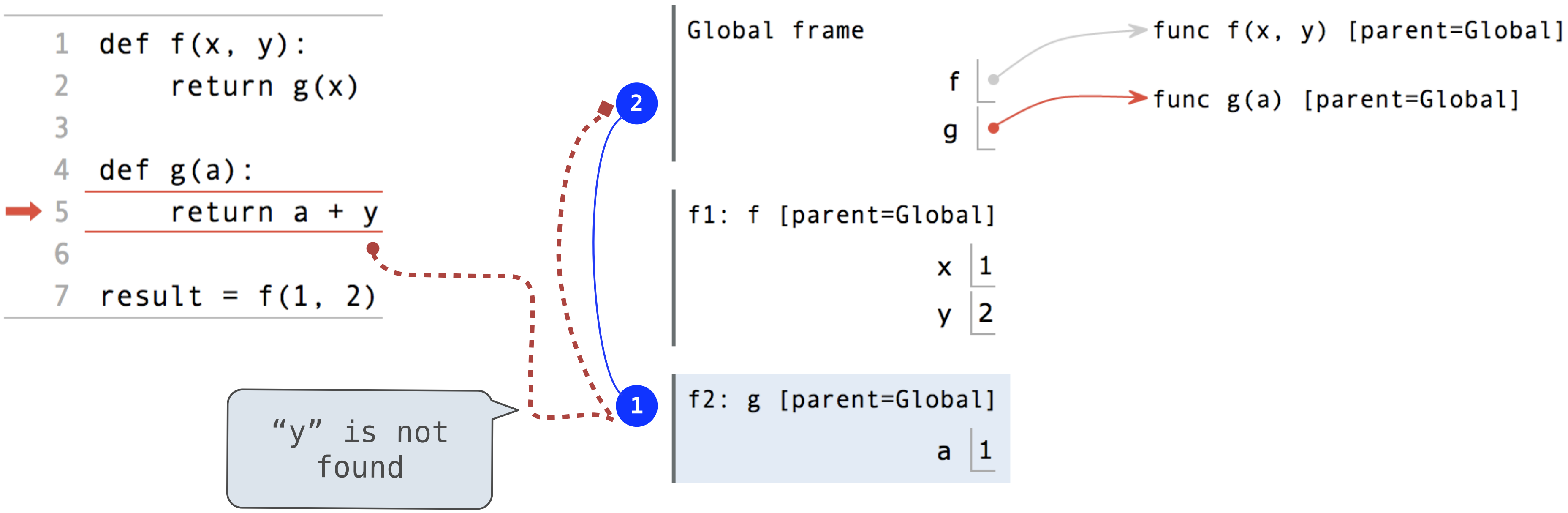
y 2

f2: g [parent=Global]

a 1

Interactive Diagram

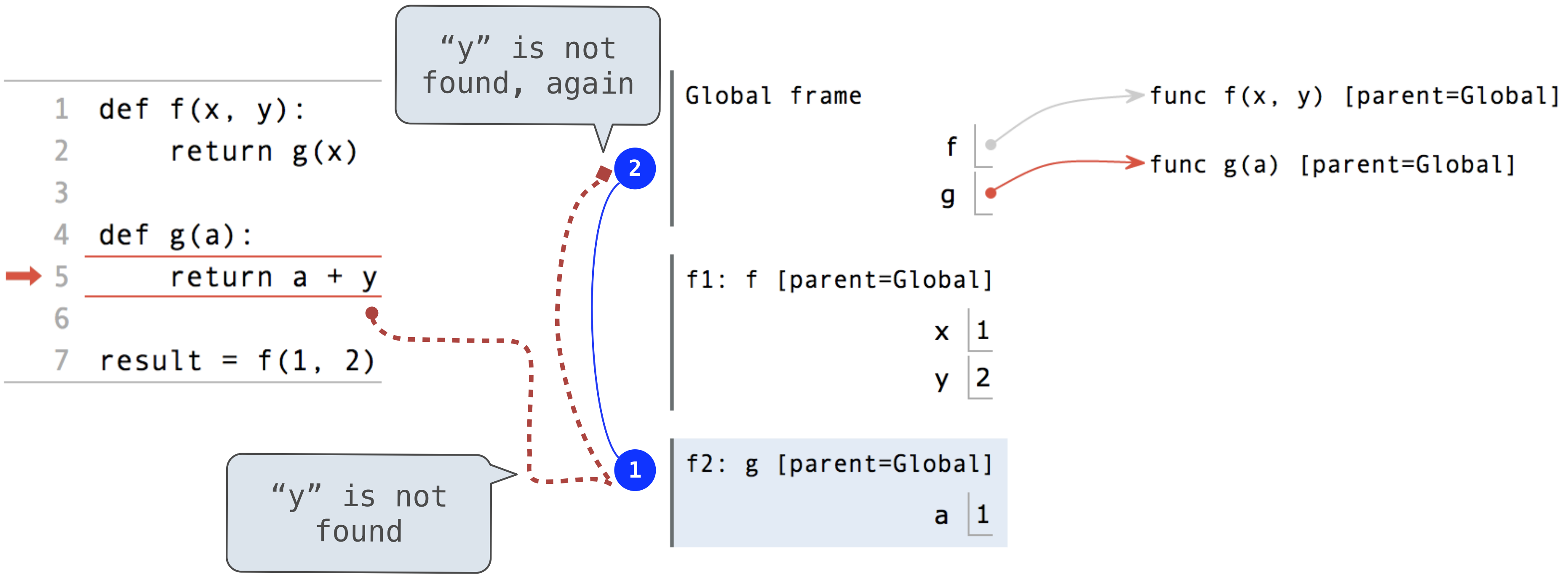
Local Names are not Visible to Other (Non-Nested) Functions



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

def g(a):
    return a + y

result = f(1, 2)
```

Interactive Diagram

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



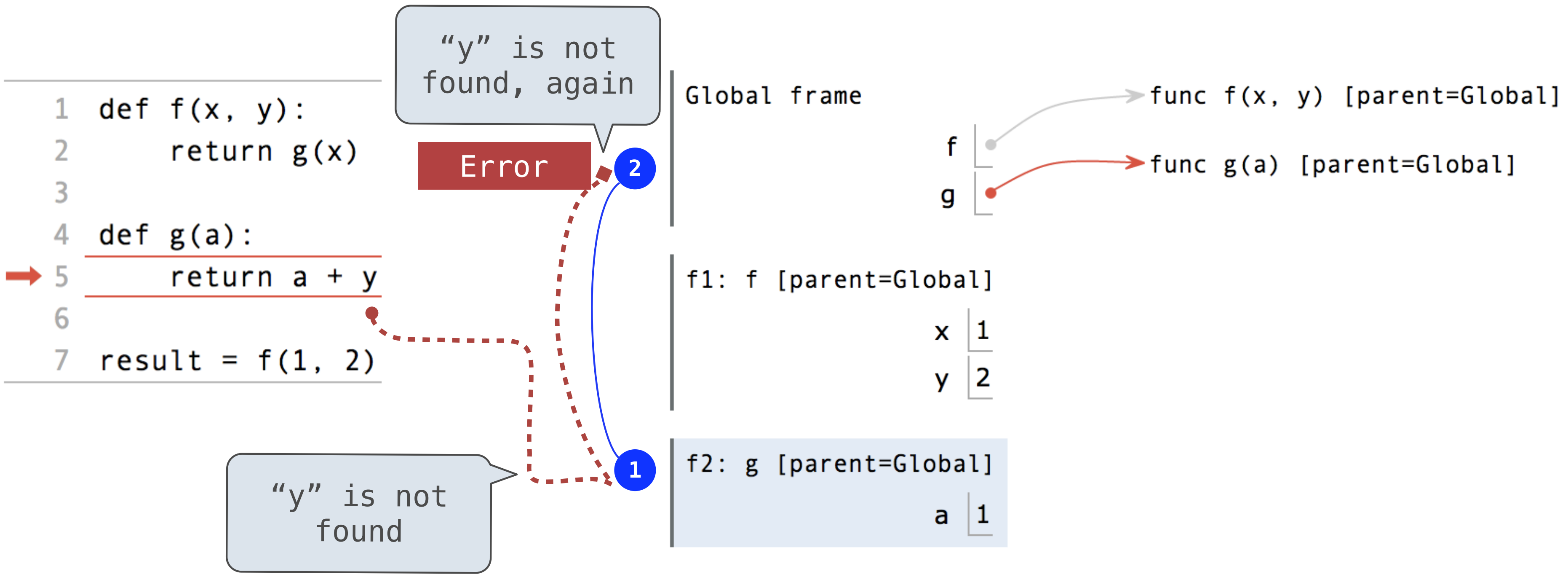
```
def f(x, y):
    return g(x)

def g(a):
    return a + y

result = f(1, 2)
```

“y” is not found, again

“y” is not found

Interactive Diagram

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)
```

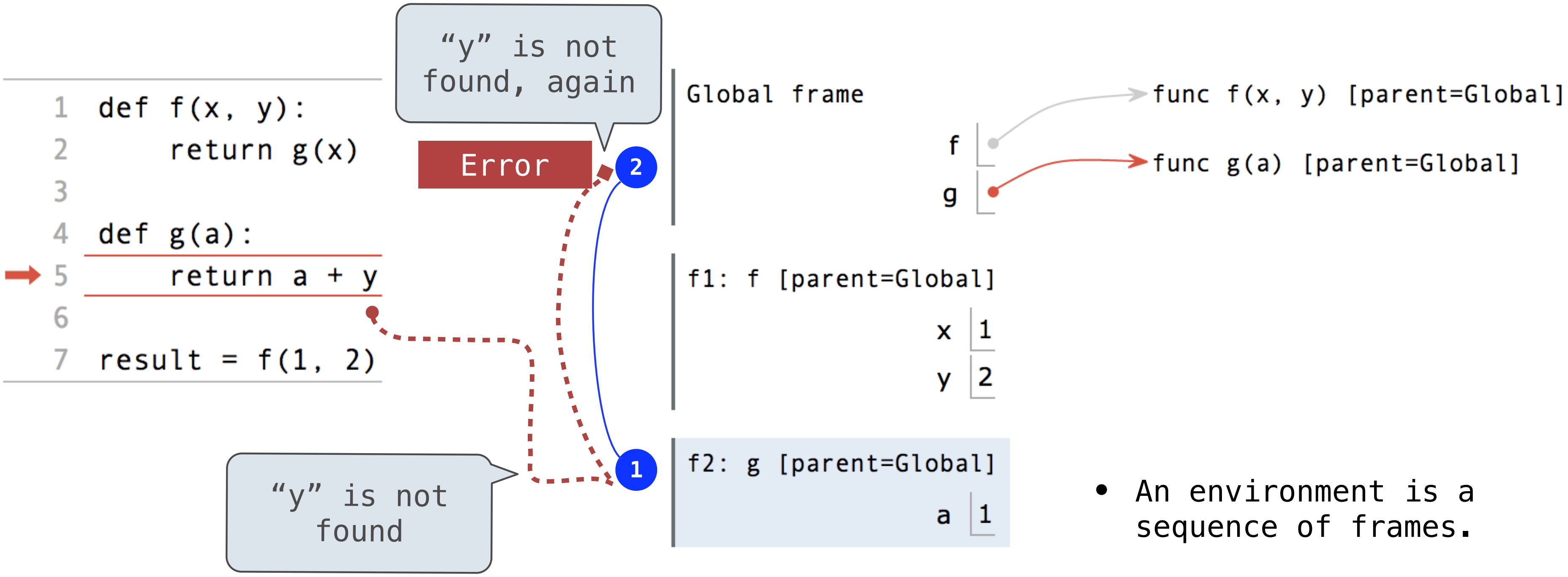
“y” is not found, again

Error

“y” is not found

Interactive Diagram

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

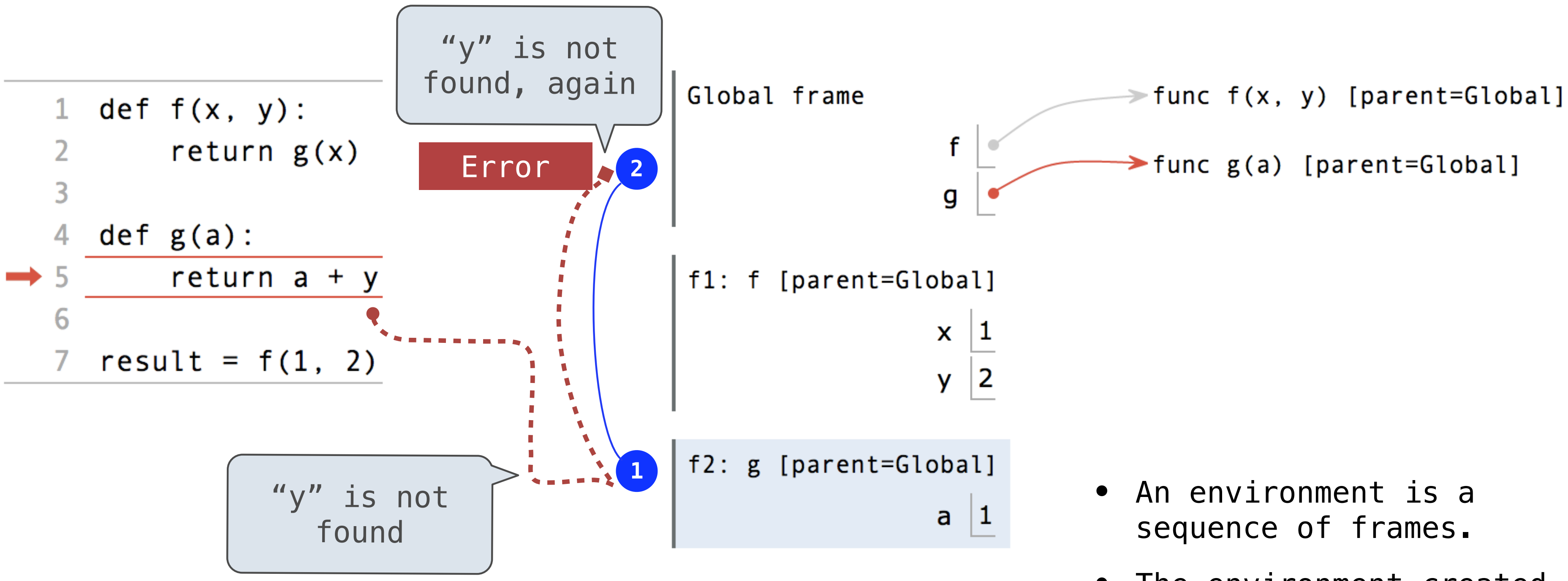


```
def f(x, y):
    return g(x)

def g(a):
    return a + y

result = f(1, 2)
```

- An environment is a sequence of frames.

Interactive Diagram

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)
```

“y” is not found, again

Error

“y” is not found

```
Global frame
                f  → func f(x, y) [parent=Global]
                g  → func g(a) [parent=Global]

f1: f [parent=Global]
                x  1
                y  2

f2: g [parent=Global]
                a  1
```

- 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.

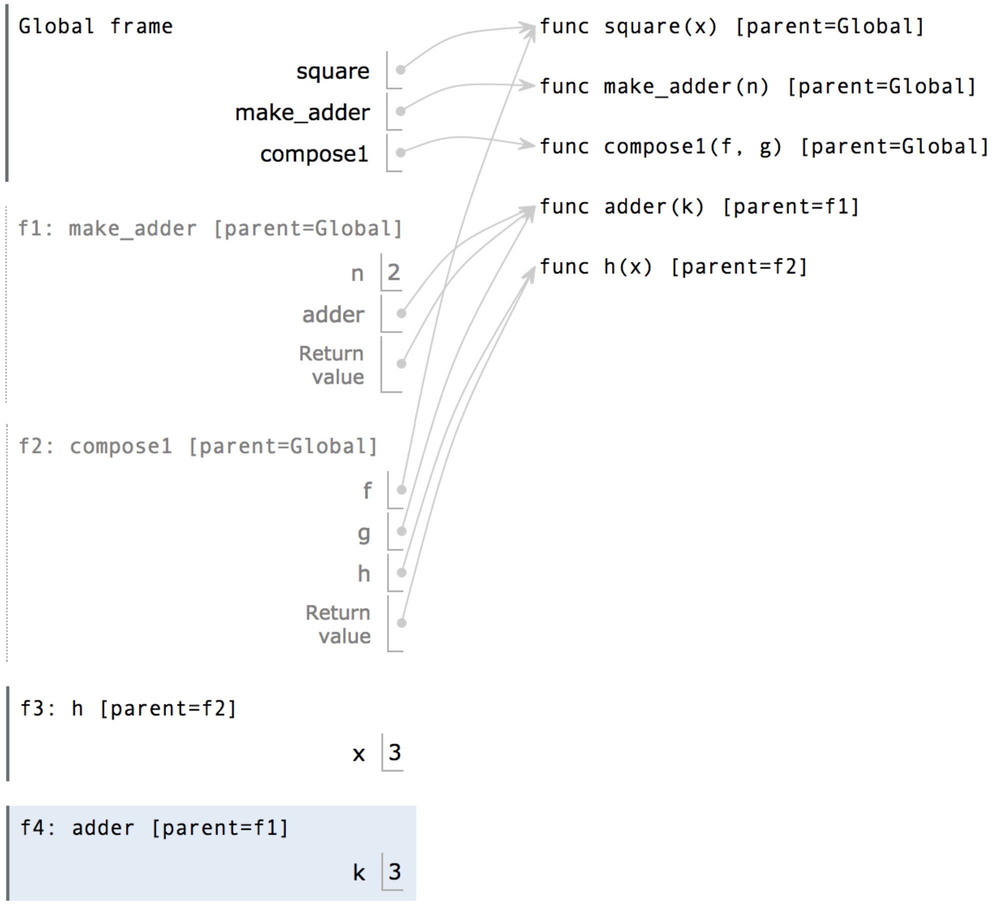
Interactive Diagram

Function Composition

(Demo)

The Environment Diagram for Function Composition

```python
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)
```

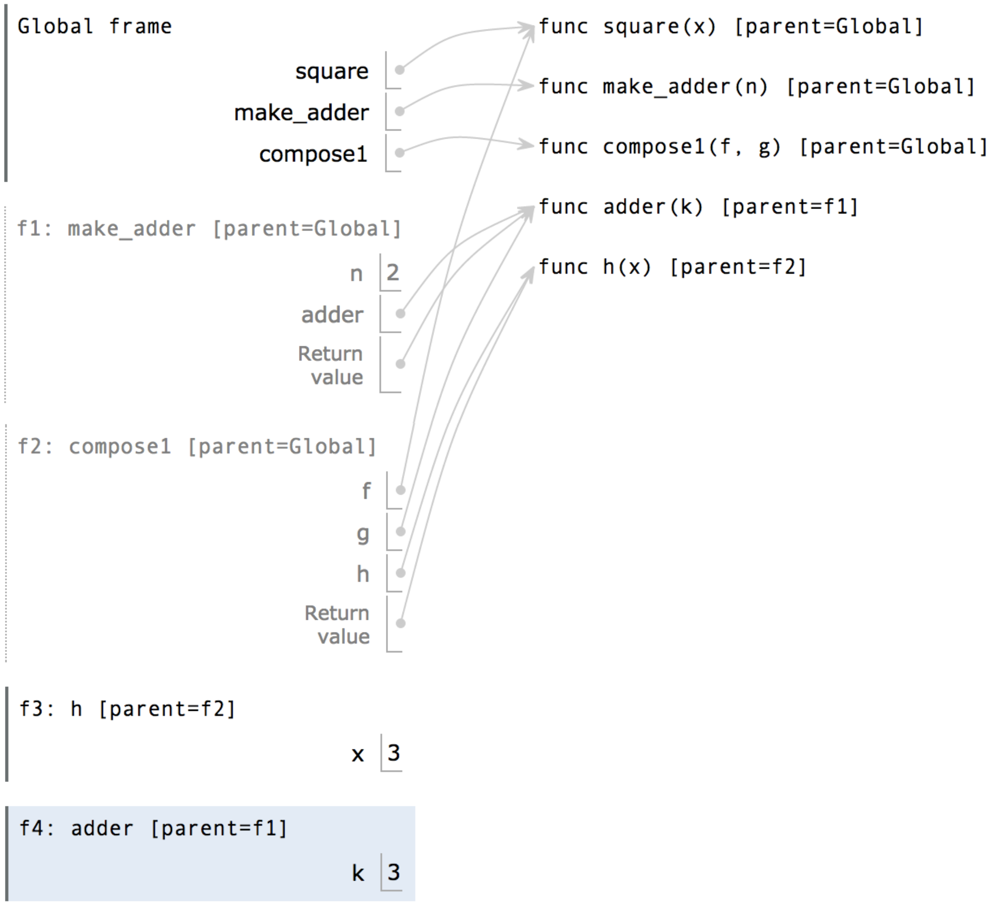


Global frame
square
make_adder
compose1
func square(x) [parent=Global]
func make_adder(n) [parent=Global]
func compose1(f, g) [parent=Global]
func adder(k) [parent=f1]
func h(x) [parent=f2]
f1: make_adder [parent=Global]
n 2
adder
Return value
f2: compose1 [parent=Global]
f
g
h
Return value
f3: h [parent=f2]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

Interactive Diagram

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)
```



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]

f1: make_adder [parent=Global]

n	2
adder	
Return value	

f2: compose1 [parent=Global]

f

g

h

Return value

f3: h [parent=f2]

x	3

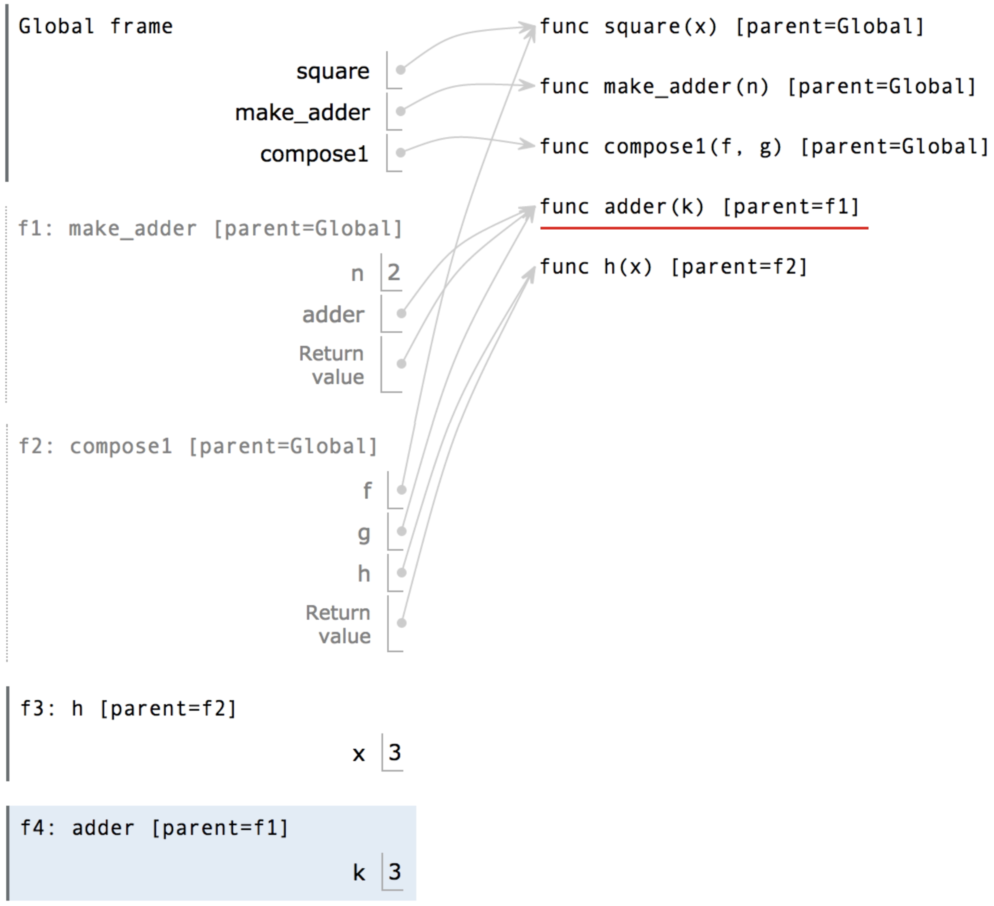
f4: adder [parent=f1]

k	3

Interactive Diagram

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)
```



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]

f1: make_adder [parent=Global]

n 2

adder

Return value

f2: compose1 [parent=Global]

f

g

h

Return value

f3: h [parent=f2]

x 3

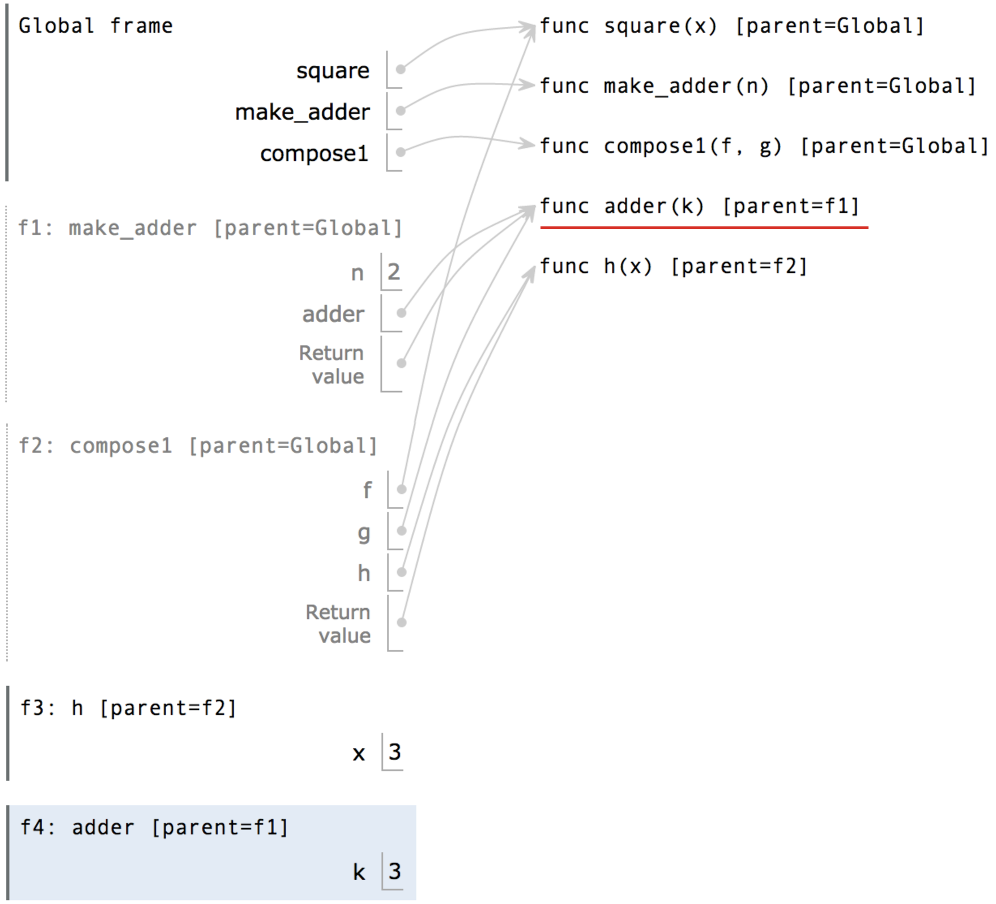
f4: adder [parent=f1]

k 3

Interactive Diagram

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)
```



Global frame

square

make_adder

compose1

f1: make_adder [parent=Global]

n 2

adder

Return value

f2: compose1 [parent=Global]

f

g

h

Return value

f3: h [parent=f2]

x 3

f4: adder [parent=f1]

k 3

func square(x) [parent=Global]

func make_adder(n) [parent=Global]

func compose1(f, g) [parent=Global]

func adder(k) [parent=f1]

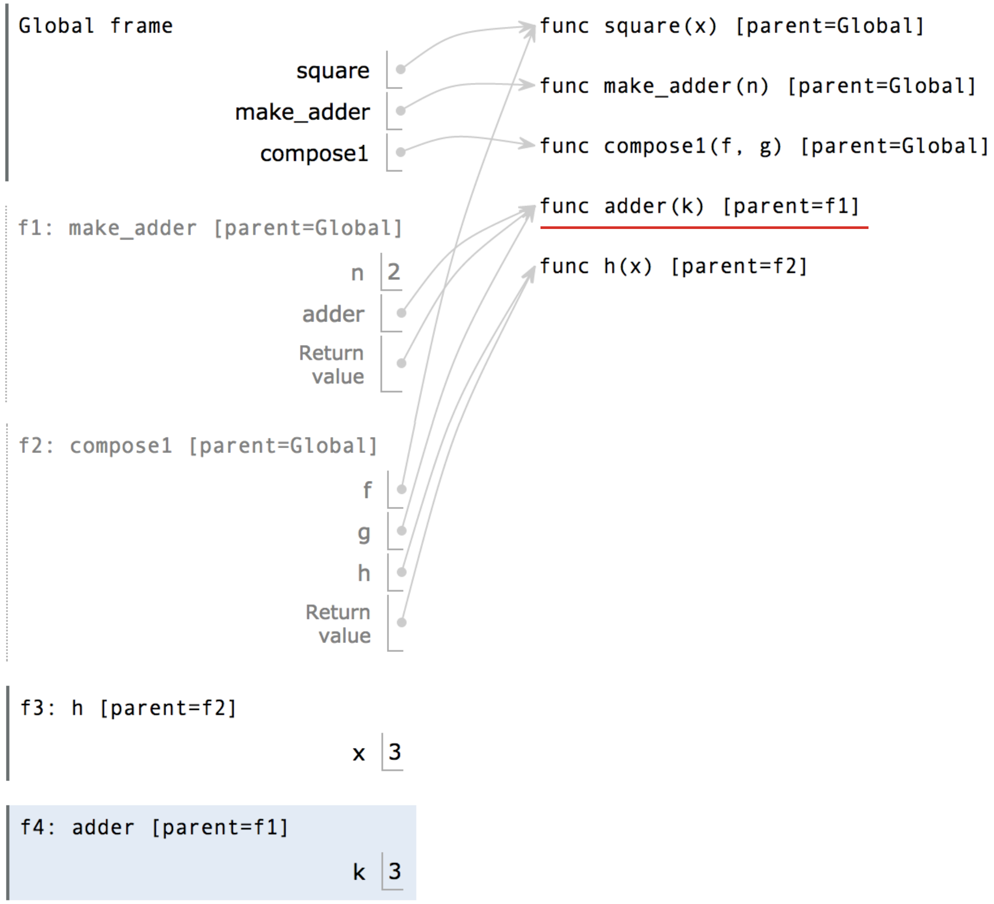
func h(x) [parent=f2]

Interactive Diagram

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



Global frame

square

make_adder

compose1

f1: make_adder [parent=Global]

n 2

adder

Return value

f2: compose1 [parent=Global]

f

g

h

Return value

f3: h [parent=f2]

x 3

f4: adder [parent=f1]

k 3

func square(x) [parent=Global]

func make_adder(n) [parent=Global]

func compose1(f, g) [parent=Global]

func adder(k) [parent=f1]

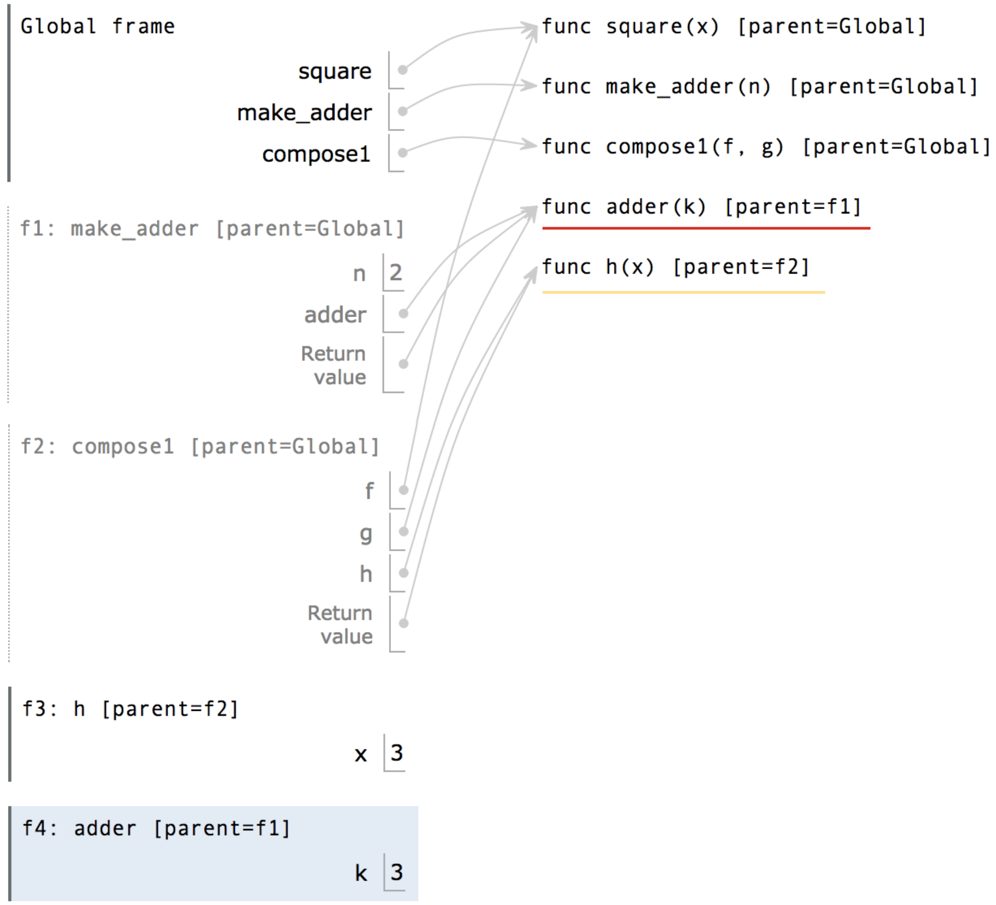
func h(x) [parent=f2]

Interactive Diagram

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



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]

f1: make_adder [parent=Global]

n 2

adder

Return value

f2: compose1 [parent=Global]

f

g

h

Return value

f3: h [parent=f2]

x 3

f4: adder [parent=f1]

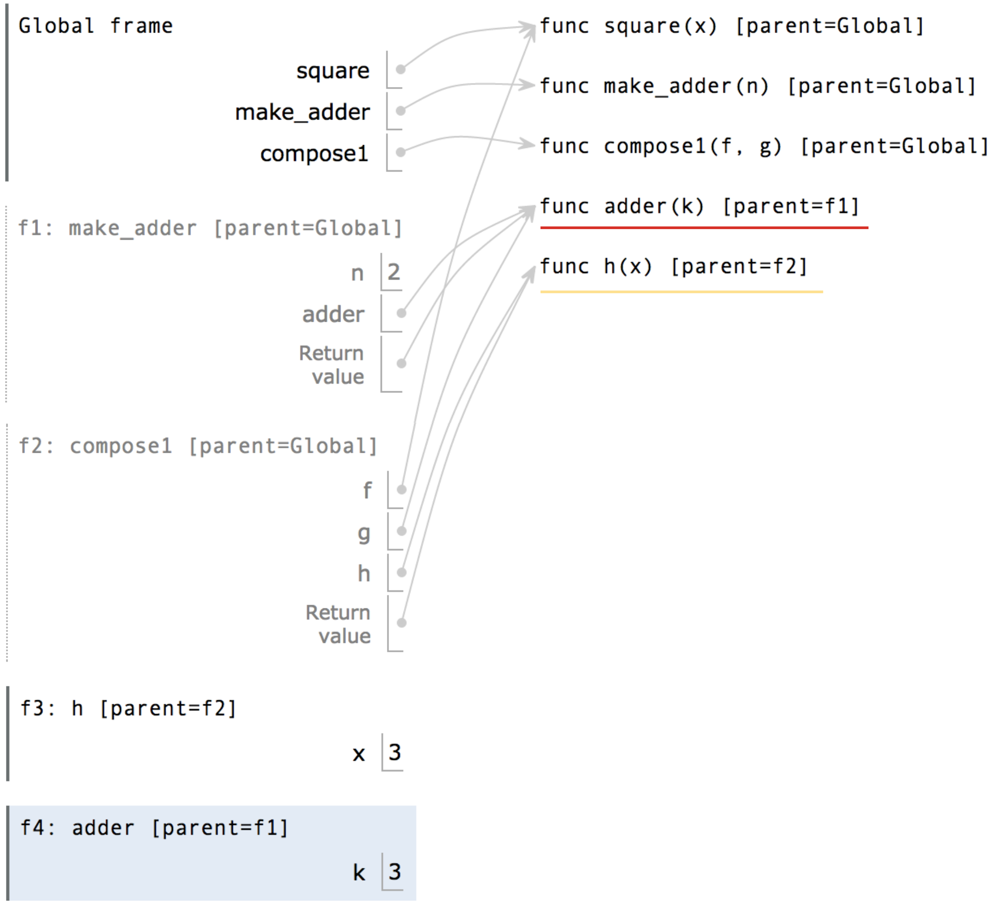
k 3

Interactive Diagram

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



Global frame

square

make_adder

compose1

f1: make_adder [parent=Global]

n 2

adder

Return value

f2: compose1 [parent=Global]

f

g

h

Return value

f3: h [parent=f2]

x 3

f4: adder [parent=f1]

k 3

func square(x) [parent=Global]

func make_adder(n) [parent=Global]

func compose1(f, g) [parent=Global]

func adder(k) [parent=f1]

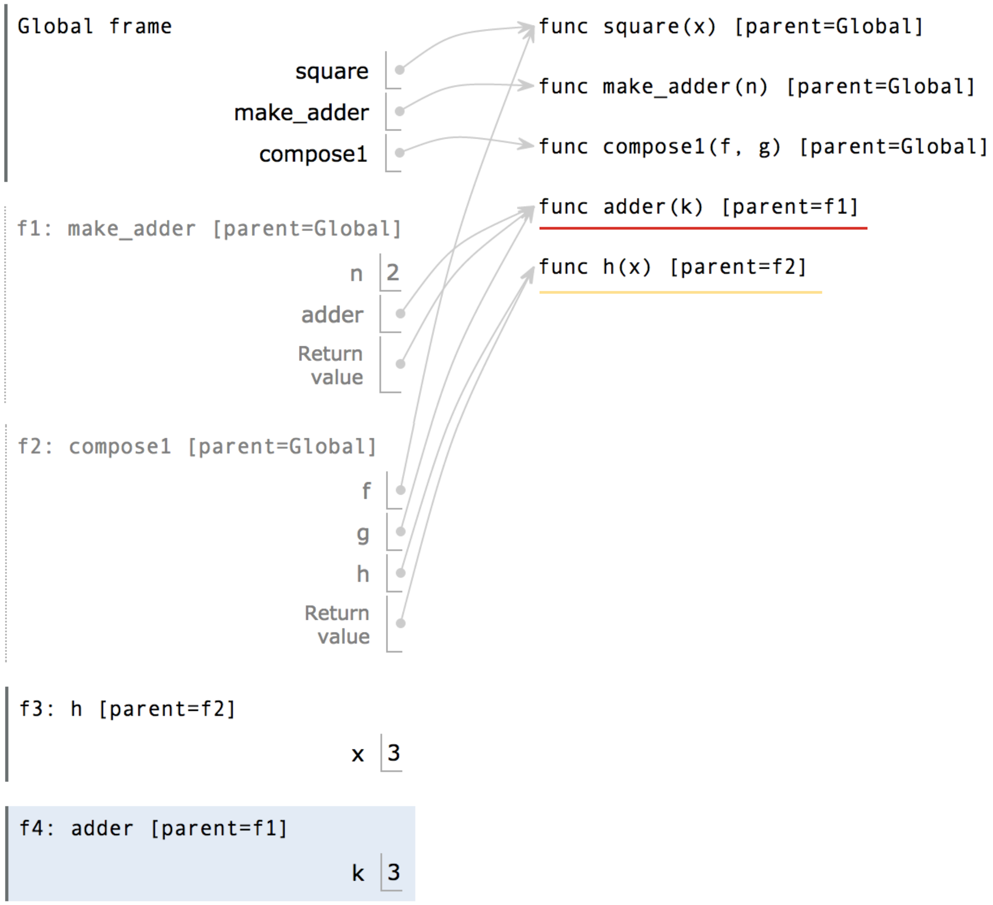
func h(x) [parent=f2]

Interactive Diagram

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



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]

f1: make_adder [parent=Global]

n 2

adder

Return value

f2: compose1 [parent=Global]

f

g

h

Return value

f3: h [parent=f2]

x 3

f4: adder [parent=f1]

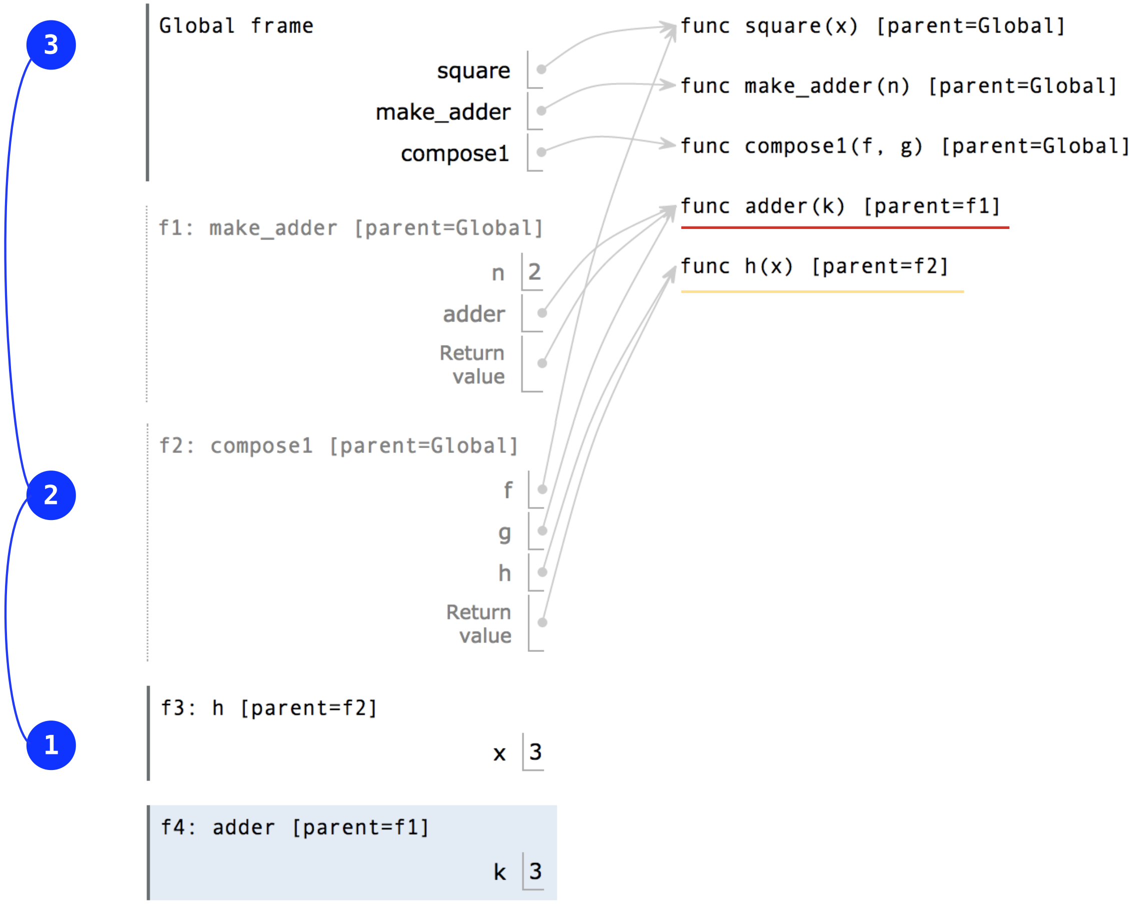
k 3

Interactive Diagram

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



Global frame
square
make_adder
compose1

f1: make_adder [parent=Global]
n 2
adder
Return value

f2: compose1 [parent=Global]
f
g
h
Return value

f3: h [parent=f2]
x 3

f4: adder [parent=f1]
k 3

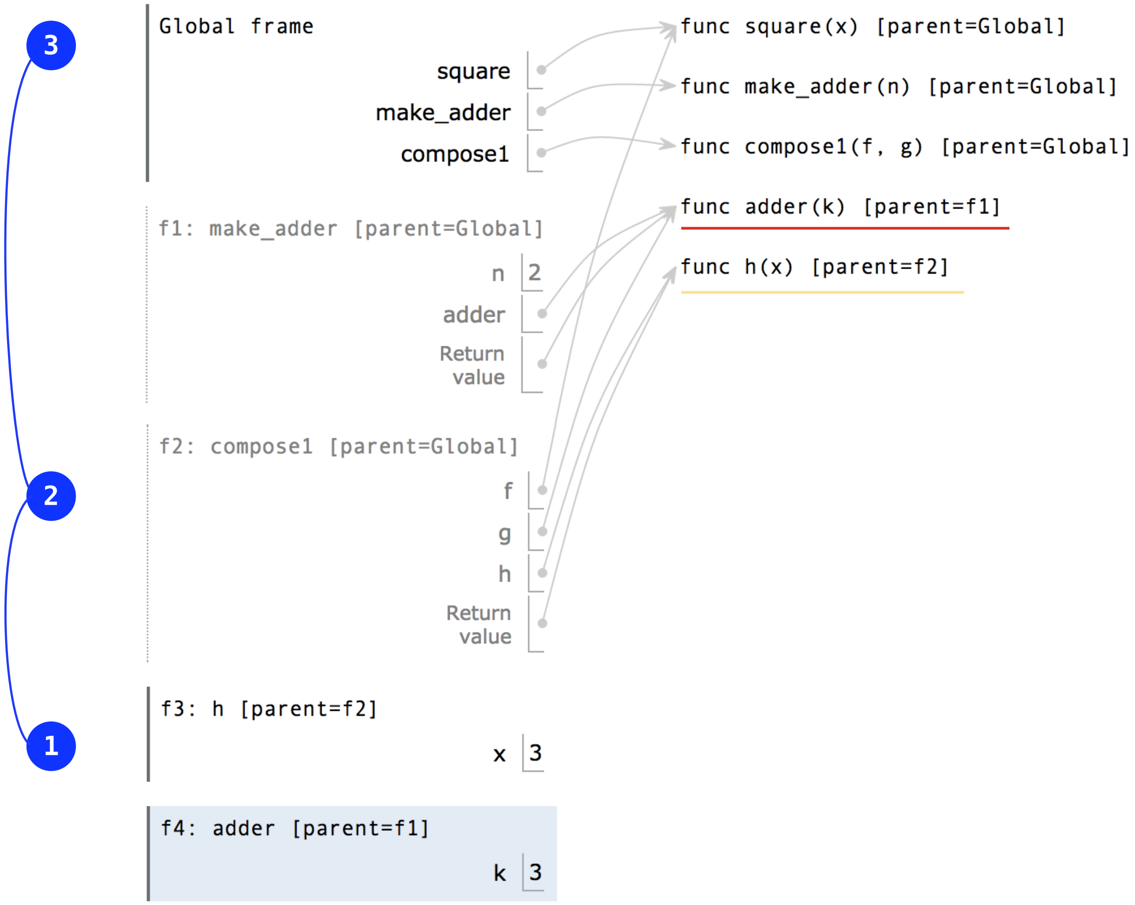
func square(x) [parent=Global]
func make_adder(n) [parent=Global]
func compose1(f, g) [parent=Global]
func adder(k) [parent=f1]
func h(x) [parent=f2]

Interactive Diagram

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



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]

f1: make_adder [parent=Global]

n 2
adder
Return value

f2: compose1 [parent=Global]

f
g
h
Return value

f3: h [parent=f2]

x 3

f4: adder [parent=f1]

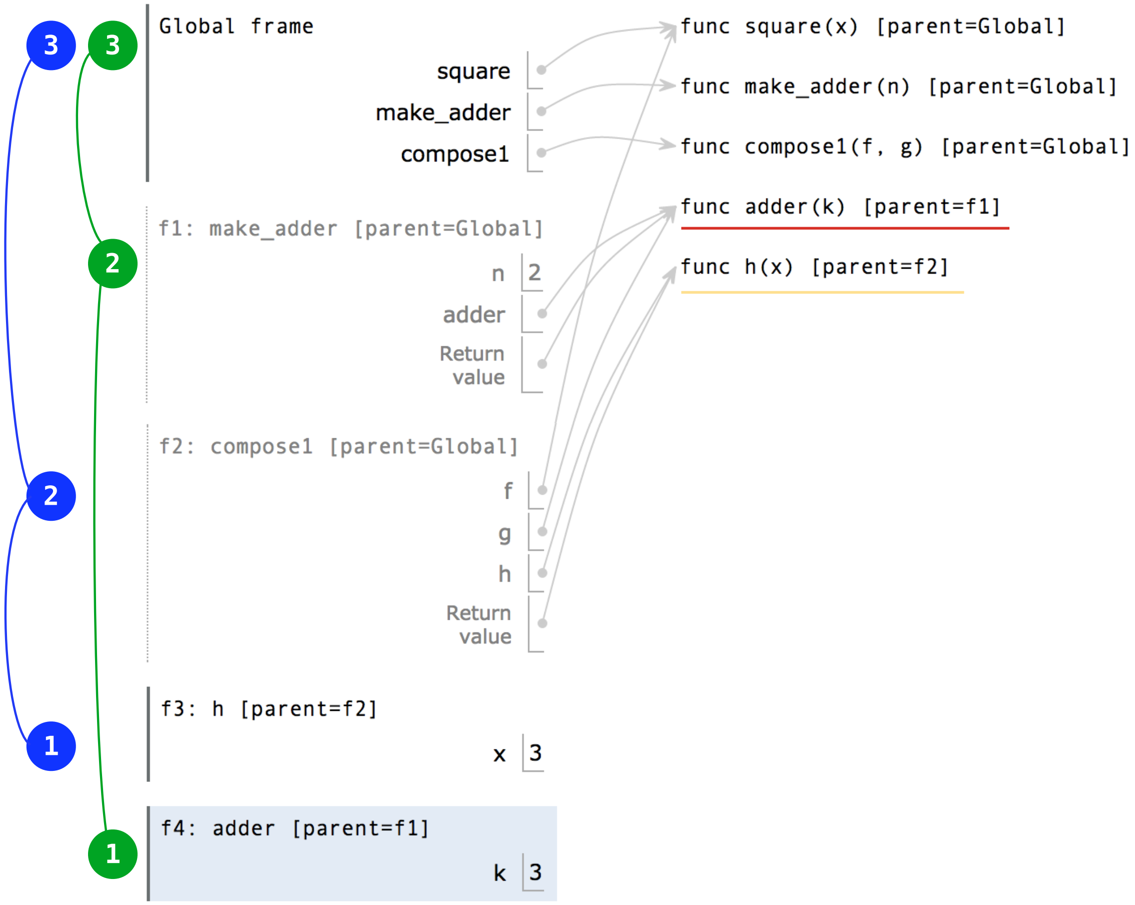
k 3

Interactive Diagram

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



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]

f1: make_adder [parent=Global]

n 2

adder

Return value

f2: compose1 [parent=Global]

f

g

h

Return value

f3: h [parent=f2]

x 3

f4: adder [parent=f1]

k 3

Interactive Diagram