61A Lecture 5



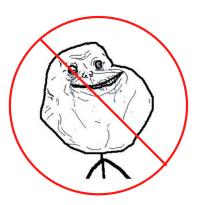
fice Hours: You Should Go!	

Office Hours: You Should Go!

You are not alone!

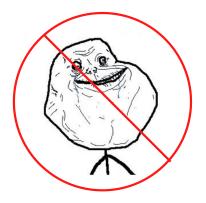
Office Hours: You Should Go!

You are not alone!



Office Hours: You Should Go!

You are not alone!



http://cs61a.org/office-hours.html

Environments for Higher-Order Functions

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

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

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!

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)

```
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)
Global frame
apply_twice
square
```

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

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

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

or func apply_
```

tutor.com/composingprograms.html#code=def%28apply_tvice%28f,%20xx29%3A%0A%20%20ereturm%20f%28r429x929v29w3A%0A%20%20ereturm%20fx20x20ereturm%20fx20x20x20ereturm%20fx20x20x20x0ereturm%20fx20x20x20x0ereturm%20fx20x20x20x0ereturm%20fx20x20x0ere

```
Global frame
                                                         func apply_twice(f, x) [parent=Global]
def apply_twice(f, x):
    return f(f(x))
                                    apply_twice
                                                         func square(x) [parent=Global]
                                        square
                                                                 Applying a user-defined function:
def square(x):
                                                                 • Create a new frame
    return x * x
                                                                 • Bind formal parameters
                                                                    (f & x) to arguments
result = apply_twice(square, 2)
                                                                 • Execute the body:
                                                                    return f(f(x))
```

curInstr=0

```
Global frame
def apply_twice(f, x):
                                                         ➤ func apply_twice(f, x) [parent=Global]
    return f(f(x))
                                    apply_twice
                                                         func square(x) [parent=Global]
                                        square
                                                                 Applying a user-defined function:
def square(x):
                                                                  • Create a new frame
    return x * x
                                                                  • Bind formal parameters
                                                                    (f & x) to arguments
result = apply_twice(square, 2)
                                                                  • 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

Gl
```

```
Global frame
def apply_twice(f, x):
                                                         func apply twice(f, x) [parent=Global]
    return f(f(x))
                                    apply_twice
                                                         func square(x) [parent=Global]
                                        square
                                                                 Applying a user-defined function:
def square(x):
                                                                 • Create a new frame
    return x * x
                                                                  • Bind formal parameters
                                                                    (f \& x) to arguments
result = apply_twice(square, 2)
                                                                 • Execute the body:
                                                                    return f(f(x))
```

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

def square(x):
    return x * x

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

func square(x) [parent=Global]

func square(x) [parent=Global]

func square(x) [parent=Global]

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

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

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

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

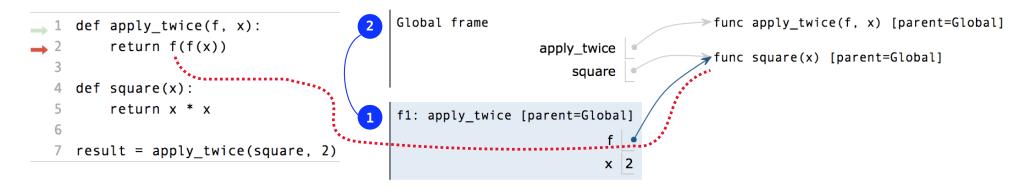
func square(x) [parent=Global]

func apply_twice(x) [parent=Global]

func apply_twice(x) [parent=Global]

func apply_twice(x) [parent=Global]
```

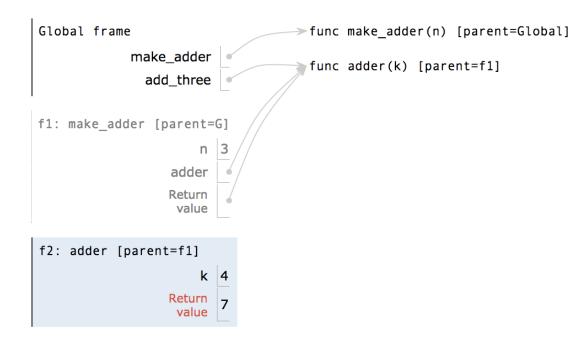
```
Global frame
def apply_twice(f, x):
                                                         ➤ func apply_twice(f, x) [parent=Global]
    return f(f(x))
                                    apply_twice
                                                         func square(x) [parent=Global]
                                         square
                                                                 Applying a user-defined function:
def square(x):
                                                                  • Create a new frame
    return x * x
                                                                  • Bind formal parameters
                                                                     (f \& x) to arguments
result = apply_twice(square, 2)
                                                                  • Execute the body:
                                                                     return f(f(x))
```



Environments for Nested Definitions

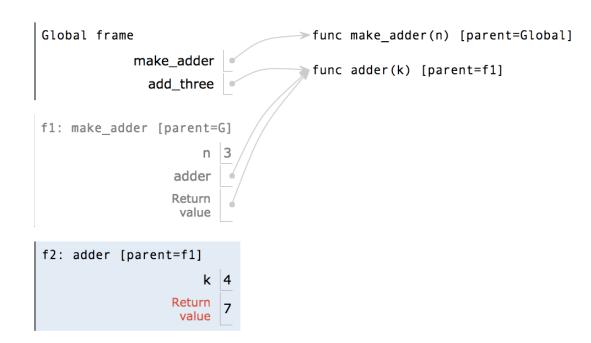
(Demo)

```
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)
```

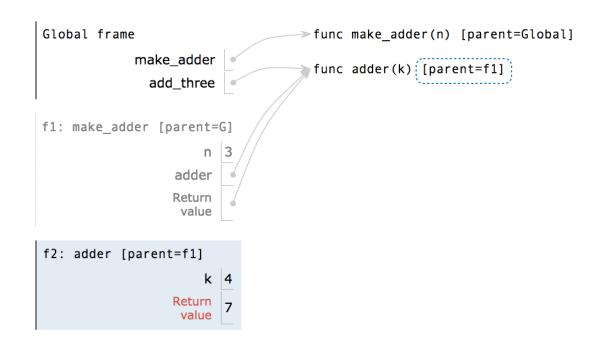


```
Nested def

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)
```

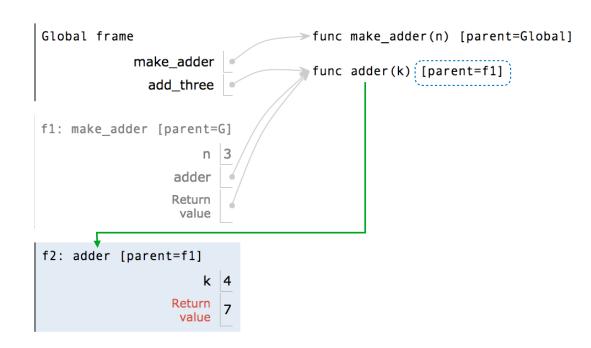


:LstJSON=[]&curInstr=0



```
Nested def

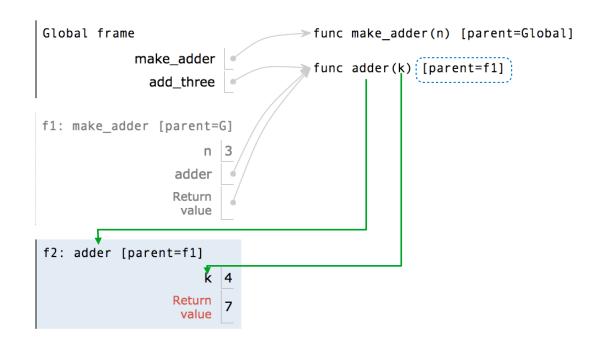
1 def make_adder(n):
2 def adder(k):
3 return k + n
4 return adder
5
6 add_three = make_adder(3)
7 add_three(4)
```



tLstJSON=[]&curInstr=0

```
Nested def

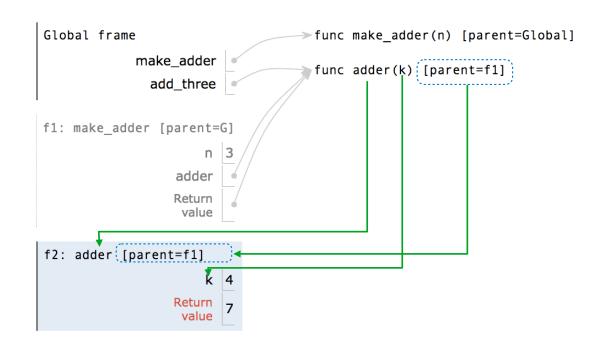
1 def make_adder(n):
2 def adder(k):
3 return k + n
4 return adder
5
6 add_three = make_adder(3)
7 add_three(4)
```



tJSON=[]&curInstr=0

```
Nested def

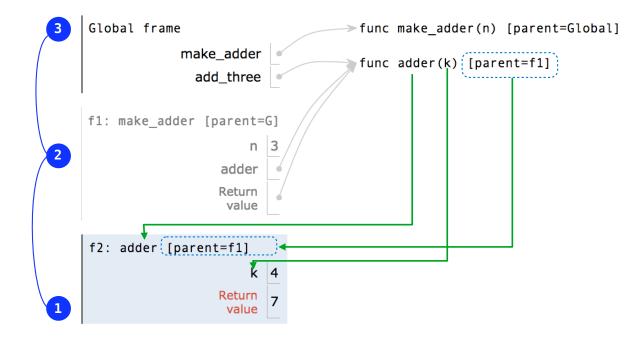
1 def make_adder(n):
2 def adder(k):
3 return k + n
4 return adder
5
6 add_three = make_adder(3)
7 add_three(4)
```



outLstJSON=[]&curInstr=0

```
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)
```



cutor.com/composingprograms.html#code=deft2@make.addert2@

```
Nested def
                                              Global frame
                                                                           > func make_adder(n) [parent=Global]
def make_adder(n):
                                                        make_adder
                                                                            func adder(k) [parent=f1]
      def adder(k):
                                                          add_three
           return k + n
                                              f1: make_adder [parent=G]
      return adder
                                                             adder
 add_three = make_adder(3)
                                                             Return
                                                             value
 add three(4)
                                              f2: adder [parent=f1]
                                                             Return
```

tutor.com/composingprograms.html&code=defe2@make_adde=r28m2993A%40A289289c2@m2dmetx28adde=r28x2993A%40A289289c2@m2dmetx15S0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36rauImputLstJS0N=||&curinstative=false&py=36ra

```
Nested def
                                                  Global frame
                                                                              > func make adder(n) [parent=Global]
     def make_adder(n):
                                                            make_adder
                                                                               func adder(k) [parent=f1]
           def adder(k):
                                                             add_three
                return k + n
                                                  f1: make_adder [parent=G]
           return adder
                                                                adder
      add_three = make_adder(3)
                                                                Return
                                                                 value
      add three(4)
                                                  f2: adder [parent=f1]
• Every user-defined function has
  a parent frame (often global)
                                                                Return
```

frame in which it was defined

```
Nested def
                                                 Global frame
                                                                             > func make adder(n) [parent=Global]
     def make_adder(n):
                                                           make_adder
                                                                              func adder(k) [parent=f1]
           def adder(k):
                                                             add_three
                return k + n
                                                 f1: make_adder [parent=G]
           return adder
                                                                adder
      add three = make adder(3)
                                                               Return
                                                                value
      add three(4)
                                                 f2: adder [parent=f1]
• Every user-defined function has
  a parent frame (often global)
                                                               Return
• The parent of a function is the
```

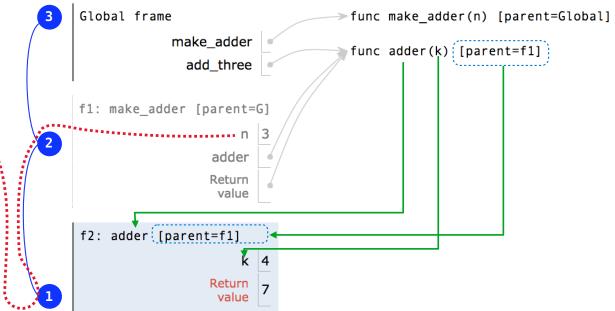
```
Nested def
                                                  Global frame
                                                                              > func make adder(n) [parent=Global]
     def make_adder(n):
                                                            make_adder
                                                                               func adder(k) [parent=f1]
           def adder(k):
                                                             add_three
                return k + n
                                                  f1: make_adder [parent=G]
           return adder
                                                                adder
      add three = make adder(3)
                                                                Return
                                                                 value
      add three(4)
                                                  f2: adder [parent=f1]
• Every user-defined function has
  a parent frame (often global)
                                                                Return
```

frame (often global)

• Every local frame has a parent

 The parent of a function is the frame in which it was defined

- 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



When a function is defined:

When a function is defined:

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

```
When a function is defined:
```

Create a function value: func <name>(<formal parameters>) [parent=<label>]
Its parent is the current frame.

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]

```
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

9

```
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:

9

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.

0

```
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>]

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.

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)

```
Global frame

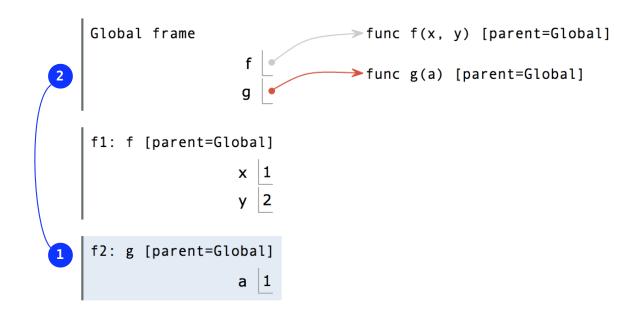
func f(x, y) [parent=Global]

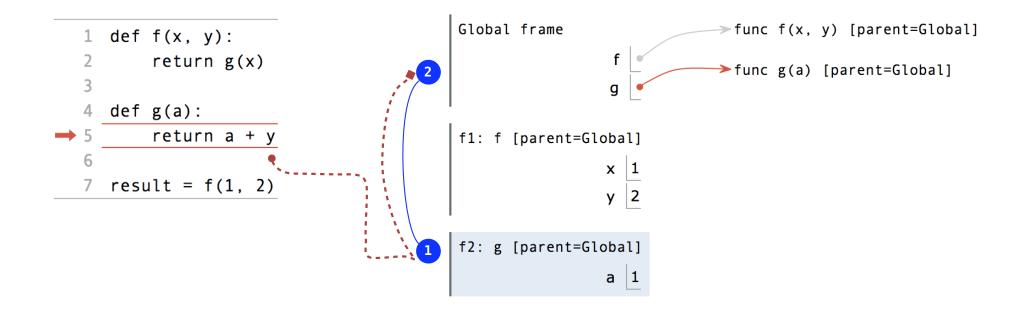
func g(a) [parent=Global]

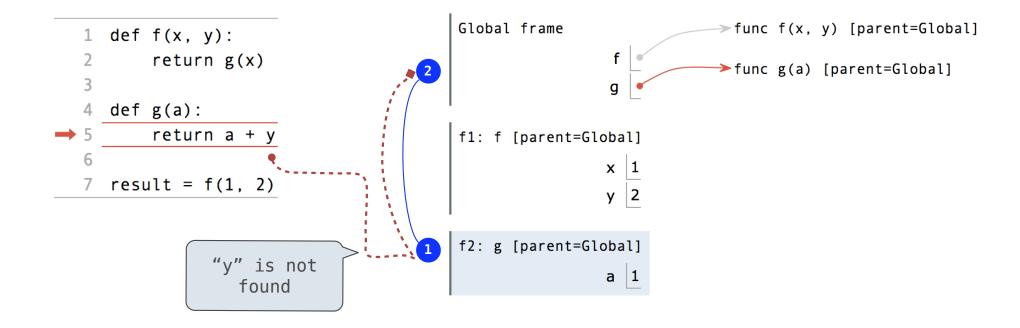
x 1
y 2

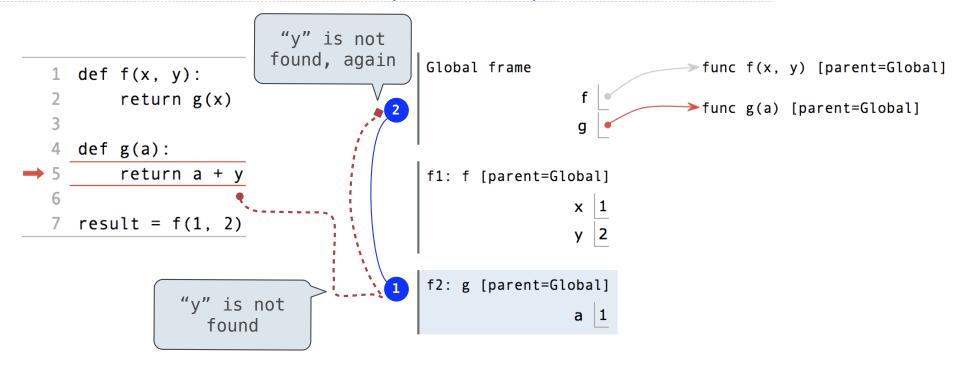
f2: g [parent=Global]

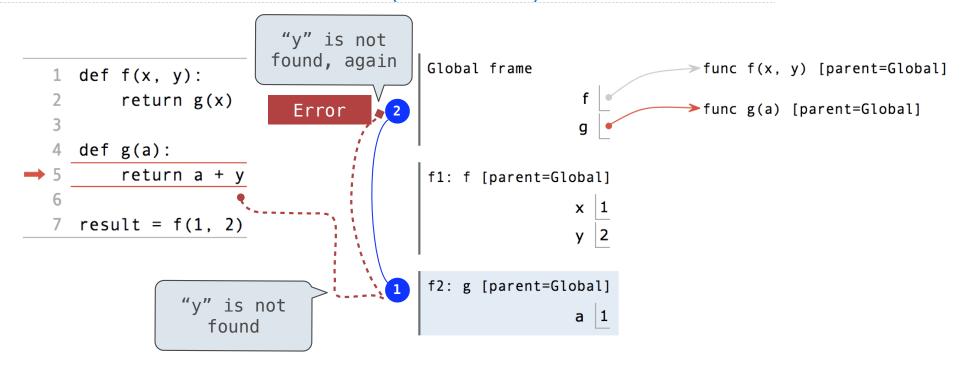
a 1
```

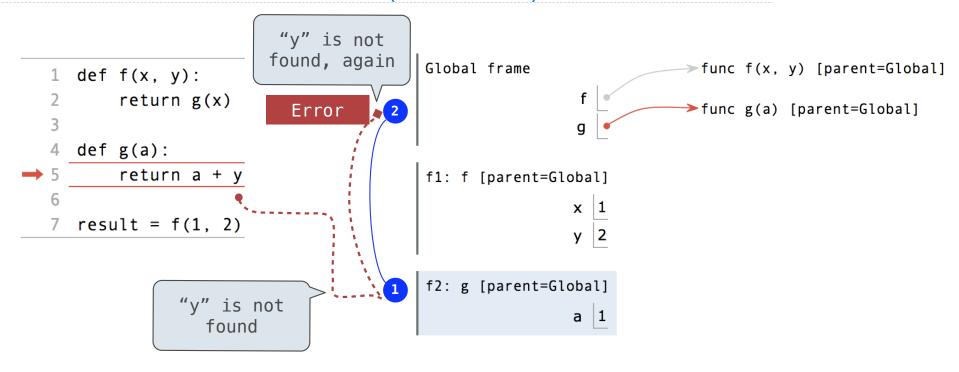




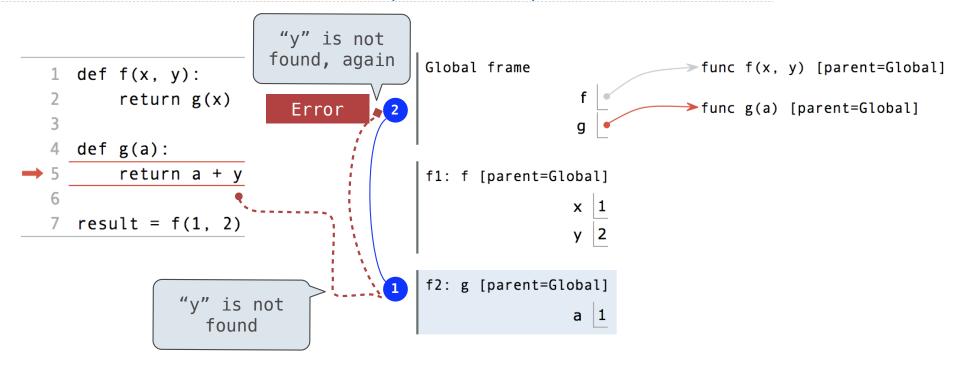








• An environment is a sequence of frames.



- 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.

Function Composition

(Demo)

```
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 composel(f, g):
10    def h(x):
11     return f(g(x))
12    return h
13
14 composel(square, make_adder(2))(3)
```

```
Global frame
                                         func square(x) [parent=Global]
                      square
                                        ►func make_adder(n) [parent=Global]
                 make_adder

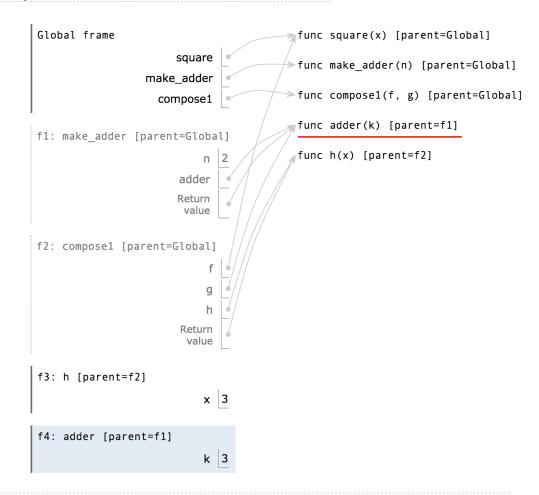
→ func compose1(f, g) [parent=Global]
                   compose1
                                         func adder(k) [parent=f1]
f1: make_adder [parent=Global]
                                         func h(x) [parent=f2]
                      adder
                      Return
                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

```
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)
```

```
Global frame
                                         func square(x) [parent=Global]
                      square
                                        ►func make_adder(n) [parent=Global]
                 make_adder

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

```
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)
```



```
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)
```

```
Global frame
                                         func square(x) [parent=Global]
                      square
                                        ►func make_adder(n) [parent=Global]
                 make_adder

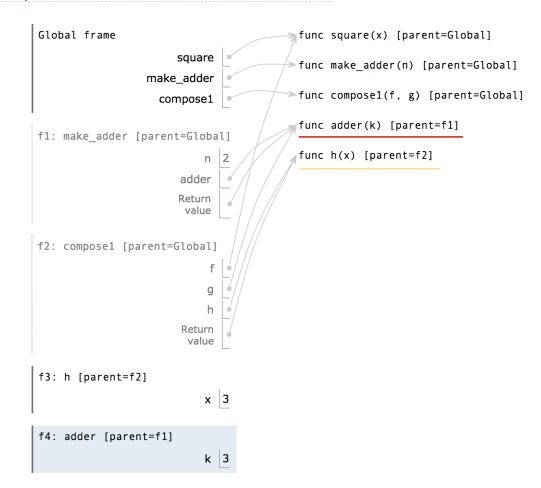
→ func compose1(f, g) [parent=Global]
                   compose1
                                         func adder(k) [parent=f1]
f1: make_adder [parent=Global]
                                         func h(x) [parent=f2]
                      adder
                      Return
                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

```
def square(x):
       return x * x
   def make adder(n):
       def adder(k):
           return k + n
       return adder
   def compose1(f, g):
10
       def h(x):
           return f(g(x))
11
12
       return h
13
  compose1(square, make_adder(2))(3)
     Return value of make_adder is
         an argument to compose1
```

```
Global frame
                                         func square(x) [parent=Global]
                      square
                                         ►func make_adder(n) [parent=Global]
                 make_adder

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

```
def square(x):
       return x * x
   def make adder(n):
       def adder(k):
           return k + n
       return adder
   def compose1(f, g):
10
       def h(x):
           return f(g(x))
11
12
       return h
13
  compose1(square, make_adder(2))(3)
     Return value of make_adder is
         an argument to compose1
```



```
def square(x):
       return x * x
   def make adder(n):
       def adder(k):
           return k + n
       return adder
   def compose1(f, g):
10
       def h(x):
           return f(g(x))
       return h
14 compose1(square, make_adder(2))(3)
     Return value of make_adder is
         an argument to compose1
```

```
Global frame
                                         func square(x) [parent=Global]
                      square
                                        ►func make adder(n) [parent=Global]
                 make_adder

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

```
def square(x):
       return x * x
   def make adder(n):
       def adder(k):
           return k + n
       return adder
   def compose1(f, g):
10
       def h(x):
           return f(g(x))
       return h
14 compose1(square, make_adder(2))(3)
     Return value of make_adder is
         an argument to compose1
```

```
Global frame
                                         func square(x) [parent=Global]
                      square
                                        ►func make adder(n) [parent=Global]
                 make_adder

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

```
Global frame
                                                                                                  func square(x) [parent=Global]
    def square(x):
                                                                                   square
         return x * x
                                                                                                  ►func make adder(n) [parent=Global]
                                                                               make_adder

→ func compose1(f, g) [parent=Global]
                                                                                compose1
    def make adder(n):
                                                                                                  func adder(k) [parent=f1]
         def adder(k):
                                                                f1: make_adder [parent=Global]
              return k + n
                                                                                                  func h(x) [parent=f2]
         return adder
                                                                                   adder
                                                                                   Return
                                                                                   value
    def compose1(f, g):
10
         def h(x):
                                                                f2: compose1 [parent=Global]
              return f(g(x))
         return h
                                                                                   Return
14 compose1(square, make_adder(2))(3)
                                                                                    value
                                                                 f3: h [parent=f2]
                                                                                      x 3
       Return value of make_adder is
           an argument to compose1
                                                                 f4: adder [parent=f1]
```

```
Global frame
                                                                                                  func square(x) [parent=Global]
    def square(x):
                                                                                  square
         return x * x
                                                                                                  ►func make adder(n) [parent=Global]
                                                                               make_adder

→ func compose1(f, g) [parent=Global]
                                                                                compose1
    def make adder(n):
                                                                                                  func adder(k) [parent=f1]
         def adder(k):
                                                                f1: make_adder [parent=Global]
              return k + n
                                                                                                  func h(x) [parent=f2]
         return adder
                                                                                   adder
                                                                                   Return
                                                                                   value
    def compose1(f, g):
10
         def h(x):
                                                                f2: compose1 [parent=Global]
              return f(g(x))
         return h
                                                                                   Return
14 compose1(square, make_adder(2))(3)
                                                                                    value
                                                                 f3: h [parent=f2]
                                                                                      x 3
       Return value of make_adder is
           an argument to compose1
                                                                 f4: adder [parent=f1]
```

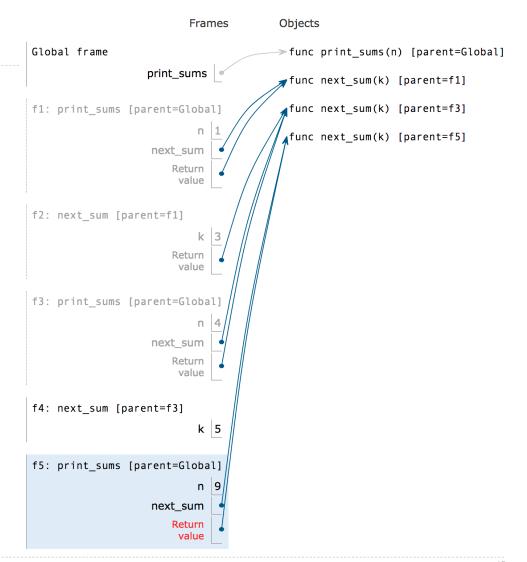
```
Global frame
                                                                                                  func square(x) [parent=Global]
    def square(x):
                                                                                   square
         return x * x
                                                                                                  ►func make_adder(n) [parent=Global]
                                                                               make_adder

→ func compose1(f, g) [parent=Global]
                                                                                compose1
    def make adder(n):
                                                                                                  func adder(k) [parent=f1]
         def adder(k):
                                                                f1: make_adder [parent=Global]
              return k + n
                                                                                                  func h(x) [parent=f2]
         return adder
                                                                                   adder
                                                                                   Return
                                                                                   value
    def compose1(f, g):
10
         def h(x):
                                                                f2: compose1 [parent=Global]
              return f(g(x))
         return h
                                                                                   Return
14 compose1(square, make_adder(2))(3)
                                                                                    value
                                                                 f3: h [parent=f2]
                                                                                      x 3
       Return value of make_adder is
           an argument to compose1
                                                                 f4: adder [parent=f1]
```

Self-Reference

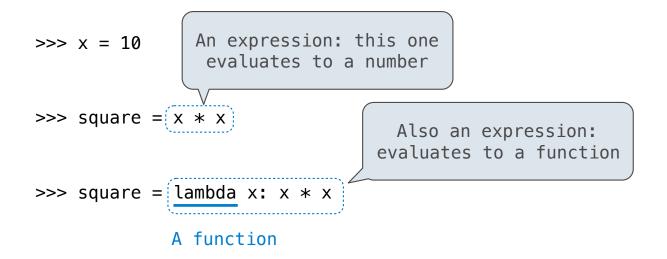
(Demo)

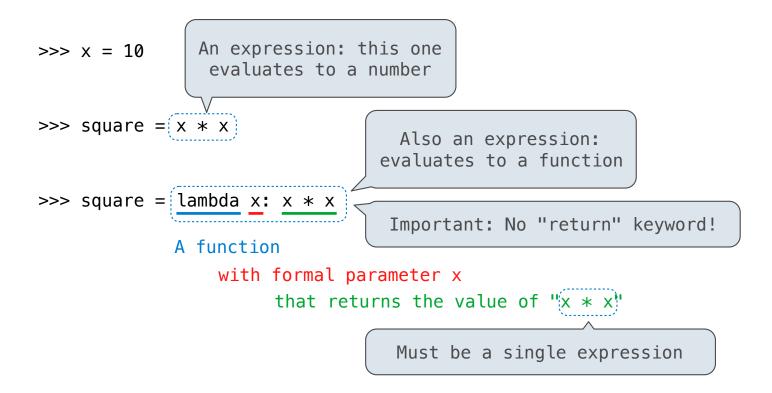
Returning a Function by Its Own Name

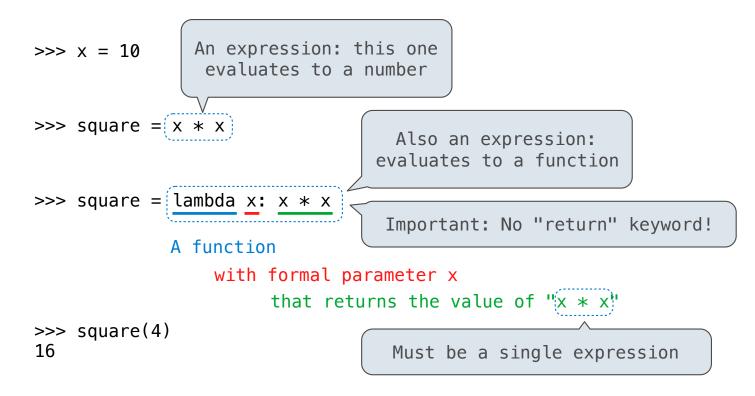


(Demo)

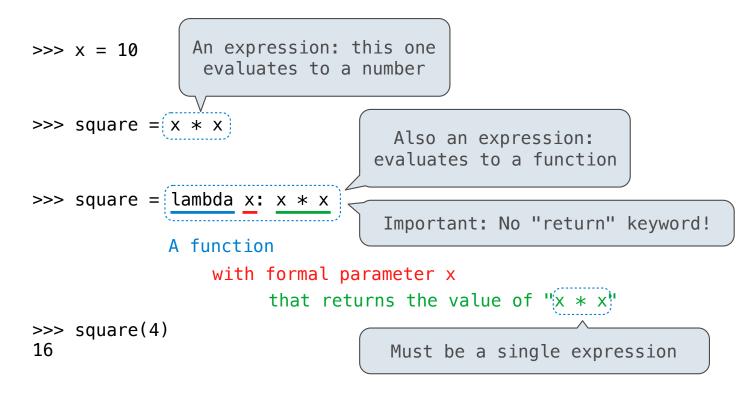
$$>>>$$
 square = $x * x$







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



Lambda expressions are not common in Python, but important in general Lambda expressions in Python cannot contain statements at all!

	Statements		

VS







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



- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.



- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.



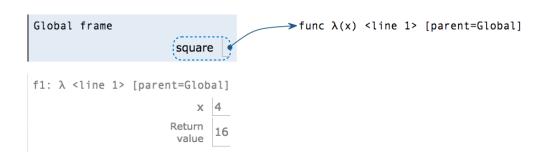
- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.



- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.

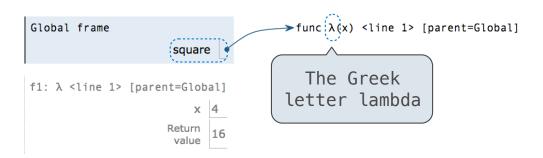


- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.



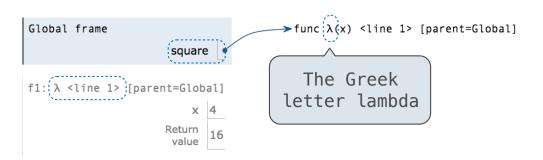


- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.





- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.





- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.

