Higher Order Functions

Thursday, 6/25

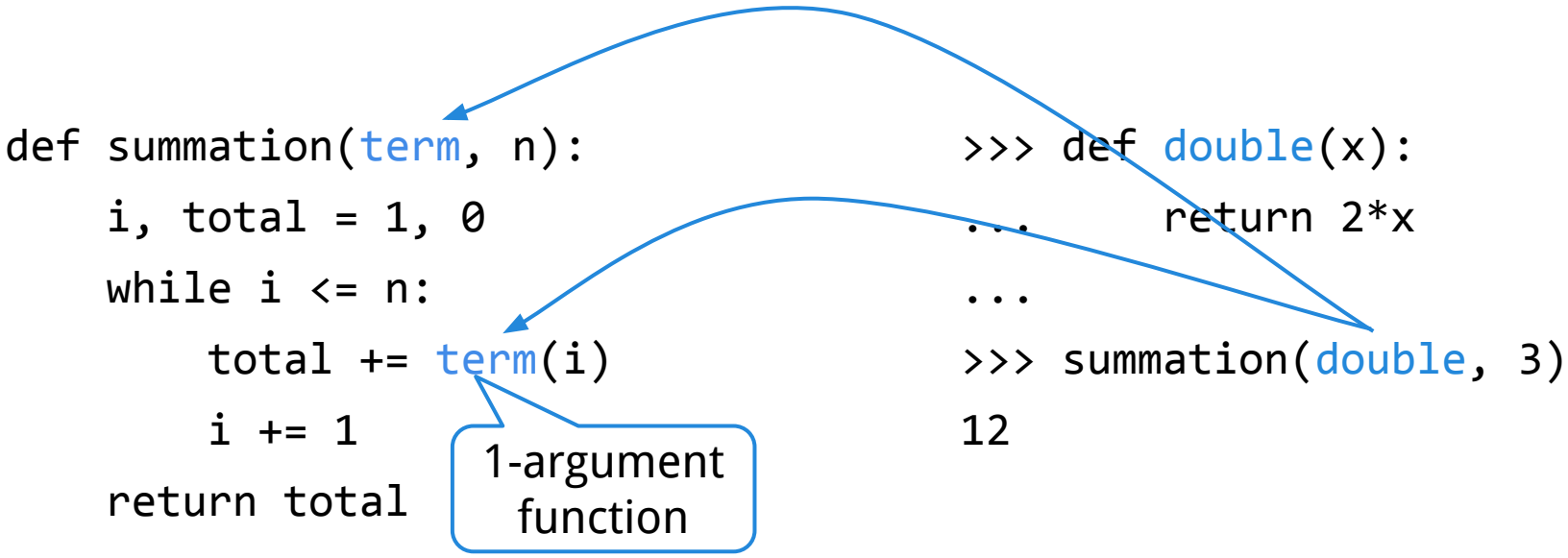
Announcements

- Fill out OK Issues survey
- Tutoring
  - Signups available today
  - Starts this Saturday
  - Tutoring sessions next week (up to Friday) will review week 1 material
- Homework 1 due Friday, June 26
- Homework 2 due Monday, June 29
- Project 1: Hog will be released this afternoon
  - Due next Thursday, July 2
  - Work with at most 1 partner
    - Must declare partner on ok!
  - Start early!

Higher Order Functions

1. Review: functions as arguments
2. Nested functions
3. Functions as return values
4. Lambda functions

Review: Functions as Arguments



Higher Order Functions

1. Review: functions as arguments
2. Nested functions
3. Functions as return values
4. Lambda functions

Nested functions

Functions can be defined inside functions.

This is useful for helper functions.

$$f(a,b,c) = \frac{ab + \frac{b}{a} + a^b}{ac + \frac{c}{a} + a^c}$$

```
def f(a, b, c):
    numer_ab = a*b + b/a + a**b
    denom_ac = a*c + c/a + a**c
    return numer_ab / denom_ac
```

DRY

numer_ab & denom_ac are very similar!

Nested functions

Helper functions can simplify code!

$$f(a,b,c) = \frac{ab + \frac{b}{a} + a^b}{ac + \frac{c}{a} + a^c}$$

How?

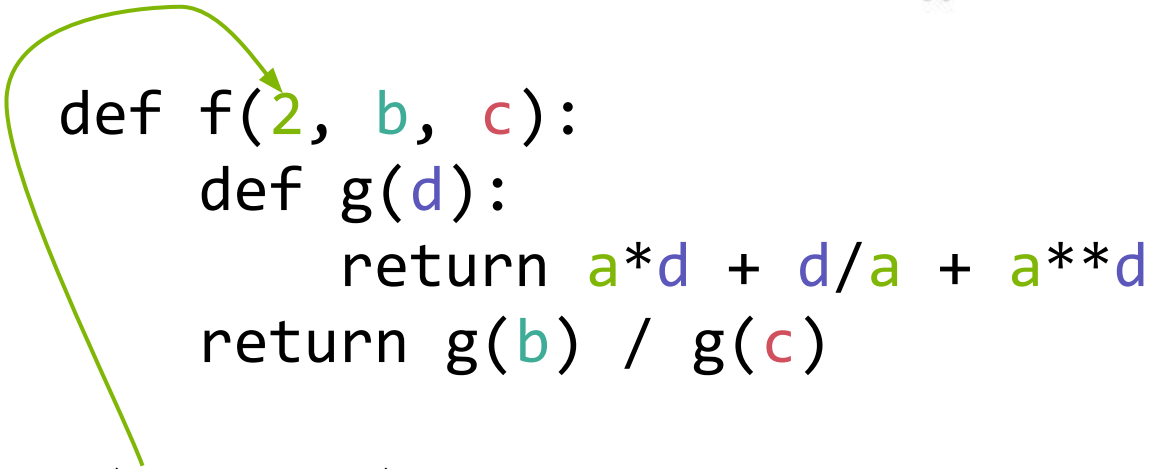
```
def f(a, b, c):
    def g(d):
        return a*d + d/a + a**d
    return g(b) / g(c)

f(2, 3, 4)
```

g is defined inside the body of f!

Nested functions

Helper functions can simplify code!

$$f(a,b,c) = \frac{ab + \frac{b}{a} + a^b}{ac + \frac{c}{a} + a^c}$$

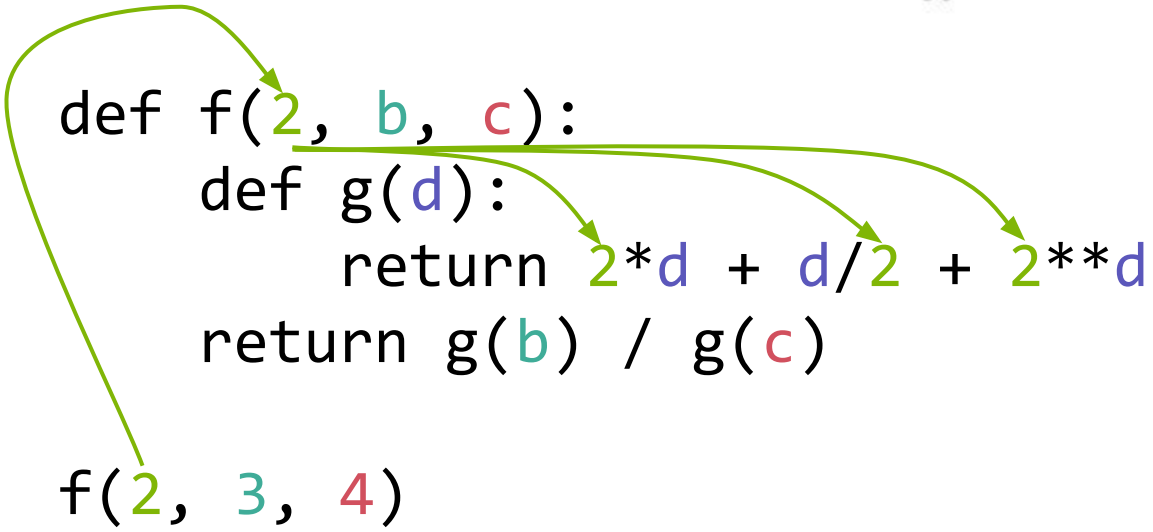


```
def f(2, b, c):
    def g(d):
        return a*d + d/a + a**d
    return g(b) / g(c)

f(2, 3, 4)
```

Nested functions

Helper functions can simplify code!

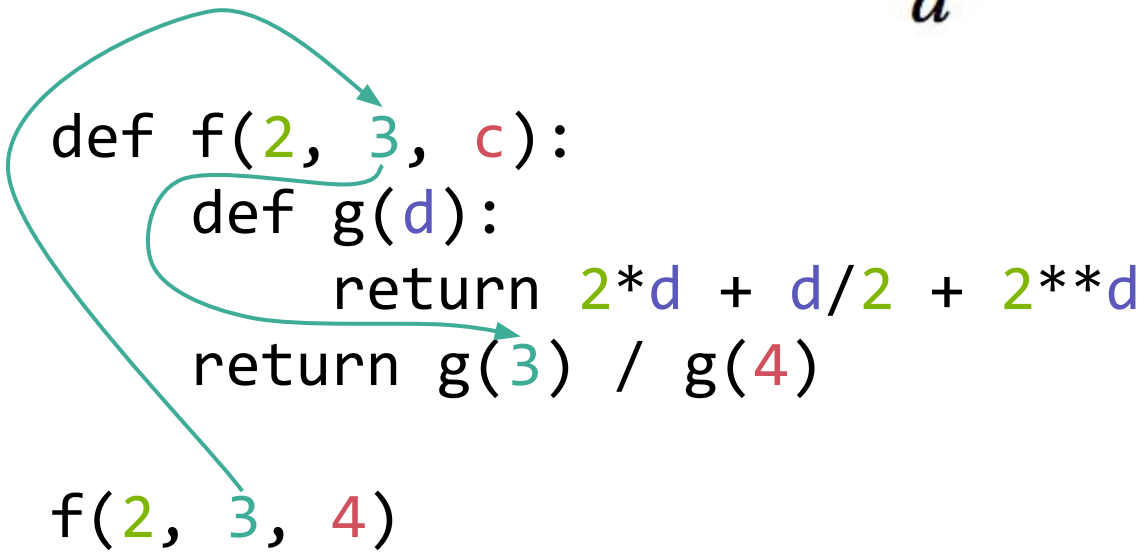
$$f(a,b,c) = \frac{ab + \frac{b}{a} + a^b}{ac + \frac{c}{a} + a^c}$$



Nested functions

Helper functions can simplify code!

$$f(a,b,c) = \frac{ab + \frac{b}{a} + a^b}{ac + \frac{c}{a} + a^c}$$

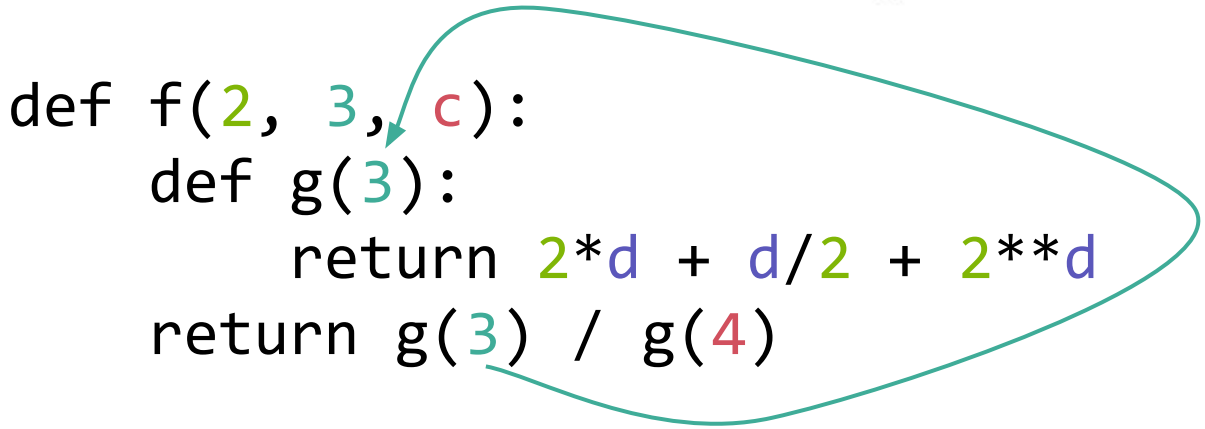


```
def f(2, 3, c):
    def g(d):
        return 2*d + d/2 + 2**d
    return g(3) / g(4)

f(2, 3, 4)
```

Nested functions

Helper functions can simplify code!

$$f(a,b,c) = \frac{ab + \frac{b}{a} + a^b}{ac + \frac{c}{a} + a^c}$$

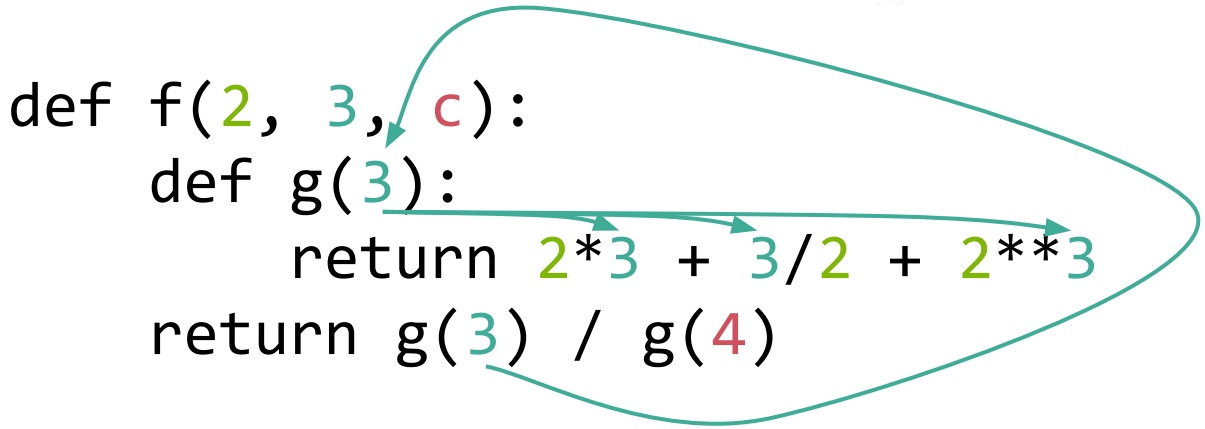


```
def f(2, 3, c):
    def g(3):
        return 2*d + d/2 + 2**d
    return g(3) / g(4)

f(2, 3, 4)
```

Nested functions

Helper functions can simplify code!

$$f(a,b,c) = \frac{ab + \frac{b}{a} + a^b}{ac + \frac{c}{a} + a^c}$$



```
def f(2, 3, c):
    def g(3):
        return 2*3 + 3/2 + 2**3
    return g(3) / g(4)

f(2, 3, 4)
```

Nested functions

Helper functions can simplify code!

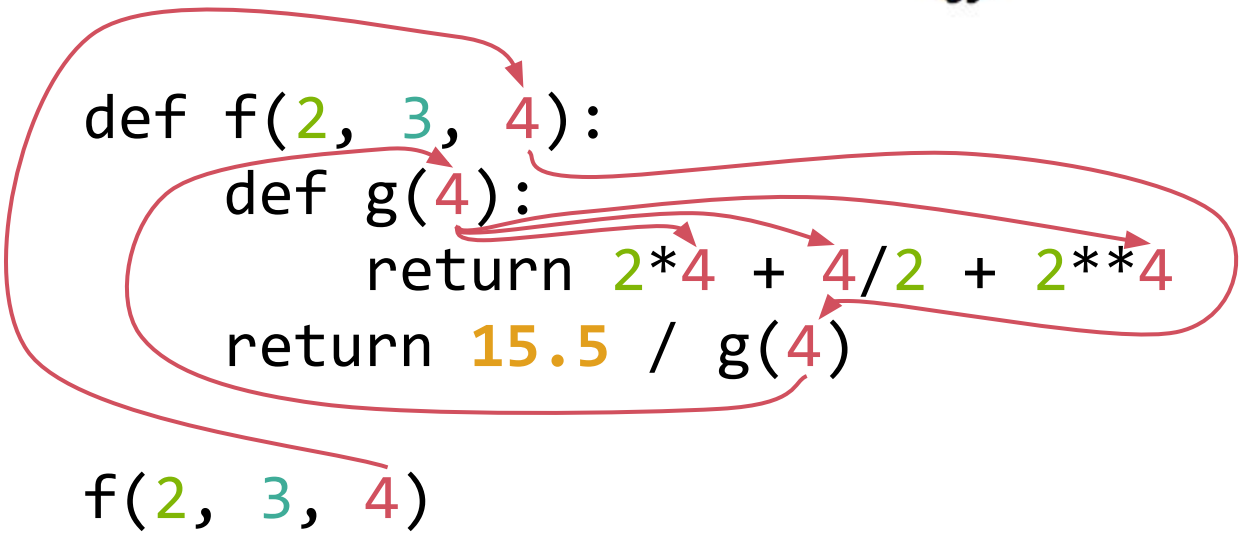
$$f(a,b,c) = \frac{ab + \frac{b}{a} + a^b}{ac + \frac{c}{a} + a^c}$$

```
def f(2, 3, c):
    def g(3):
        return 2*3 + 3/2 + 2**3
    return 15.5 / g(4)

f(2, 3, 4)
```

Nested functions

Helper functions can simplify code!

$$f(a,b,c) = \frac{ab + \frac{b}{a} + a^b}{ac + \frac{c}{a} + a^c}$$



Nested functions

Helper functions can simplify code!

$$f(a,b,c) = \frac{ab + \frac{b}{a} + a^b}{ac + \frac{c}{a} + a^c}$$

```
def f(2, 3, 4):
    def g(4):
        return 2*4 + 4/2 + 2**4
    return 15.5 / 26.0

f(2, 3, 4)
```

Nested functions

Helper functions can simplify code!

$$f(a,b,c)=\frac{ab+\frac{b}{a}+a^b}{ac+\frac{c}{a}+a^c}$$

```
def f(2, 3, 4):
    def g(4):
        return 2*4 + 4/2 + 2**4
    return 15.5 / 26.0
```

```
1.6774193548387097
```

Higher Order Functions

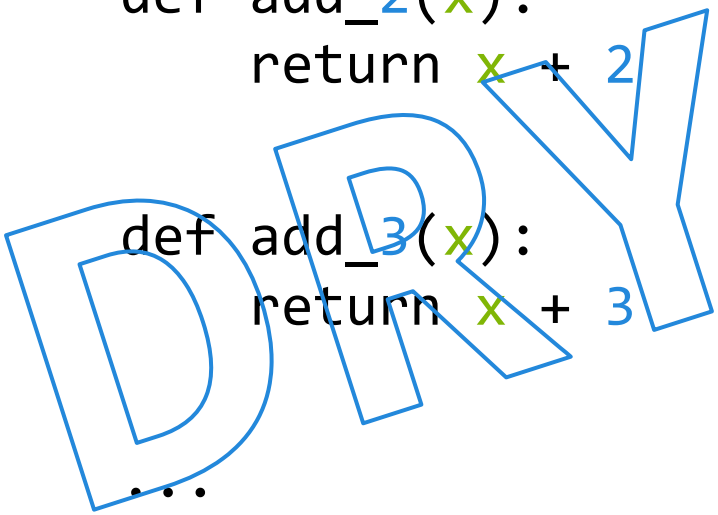
1. Review: functions as arguments
2. Nested functions
3. Functions as return values
4. Lambda functions

Adders

Functions that add some number to another number

(Demo)

```
def add_1(x):
    return x + 1

def add_2(x):
    return x + 2

def add_3(x):
    return x + 3
...

def add_128(x):
    return x + 128
```



make_adder

function that returns a function

```
def make_adder(n):
    """Return func that takes one arg k and returns k + n

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

name add_three is bound to a function

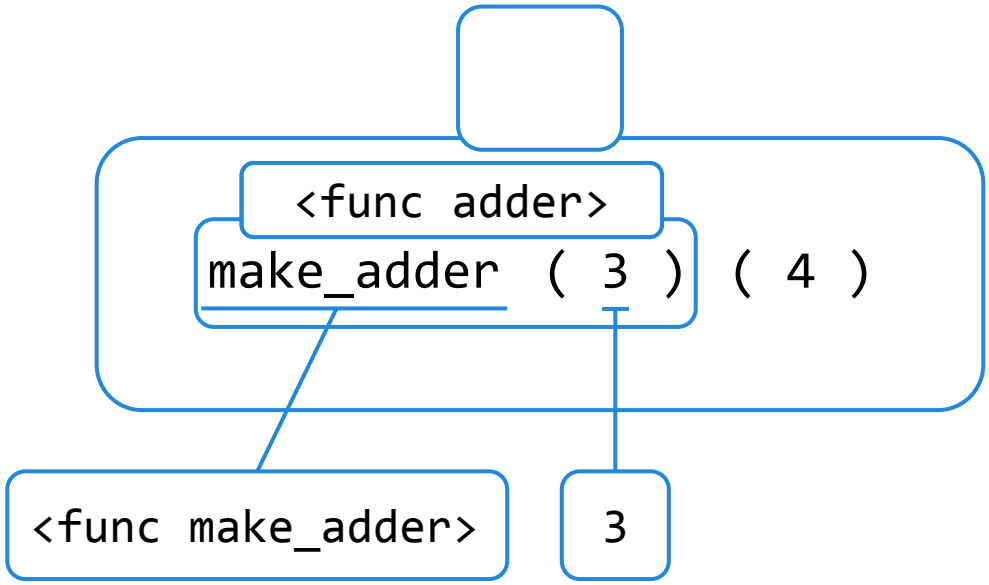
nested function

can refer to names in enclosing functions

make_adder

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

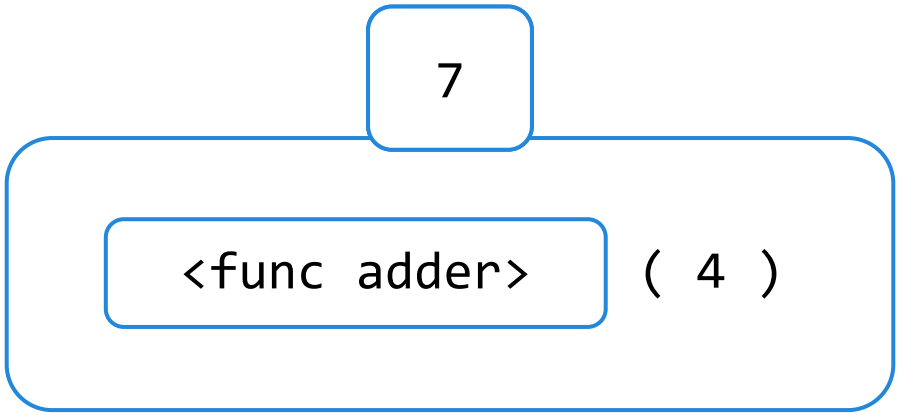
For compound call expressions, read left to right



make_adder

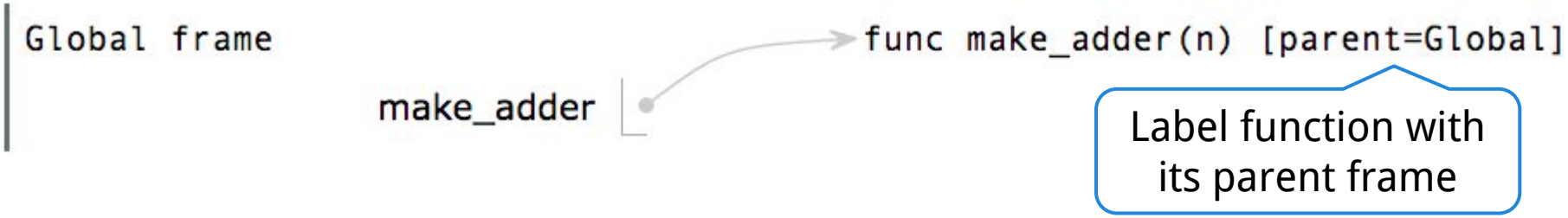
```
def make_adder(n):
    def adder(k):
        return k + n
    return adder
```

For compound call expressions, read left to right



Environment diagram: make_adder

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

make_adder(3)(4)
```

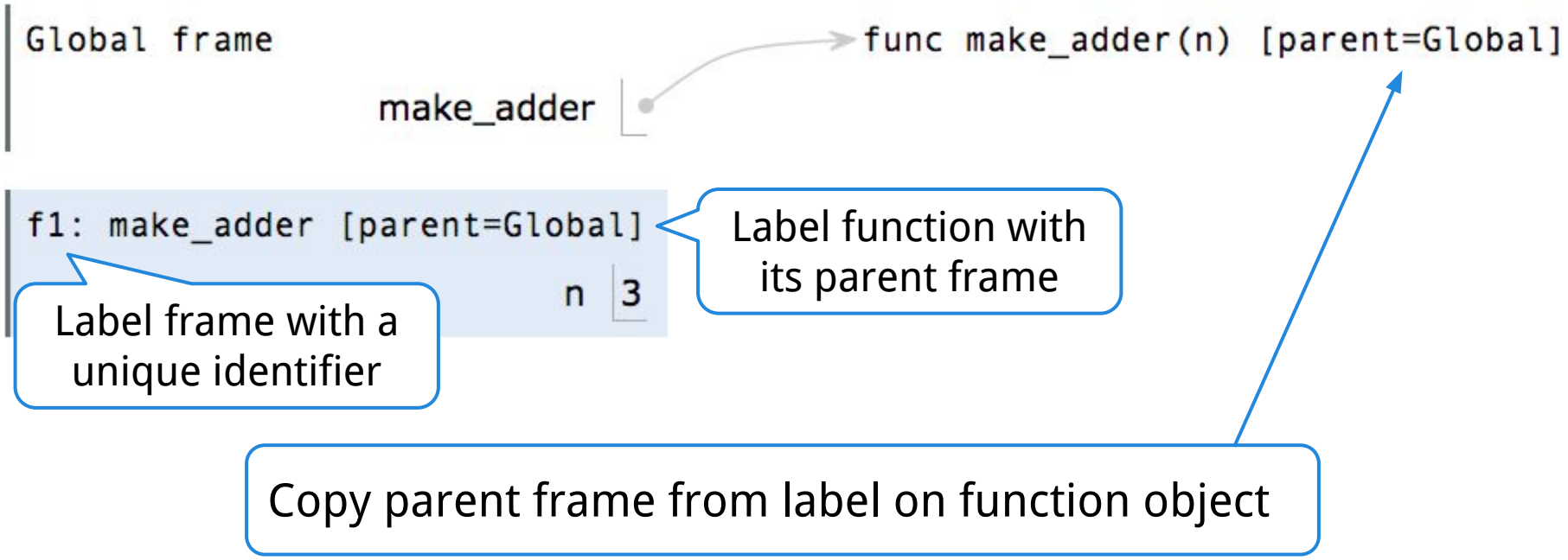


Parent frame: frame in which the function is defined

Environment diagram: make_adder

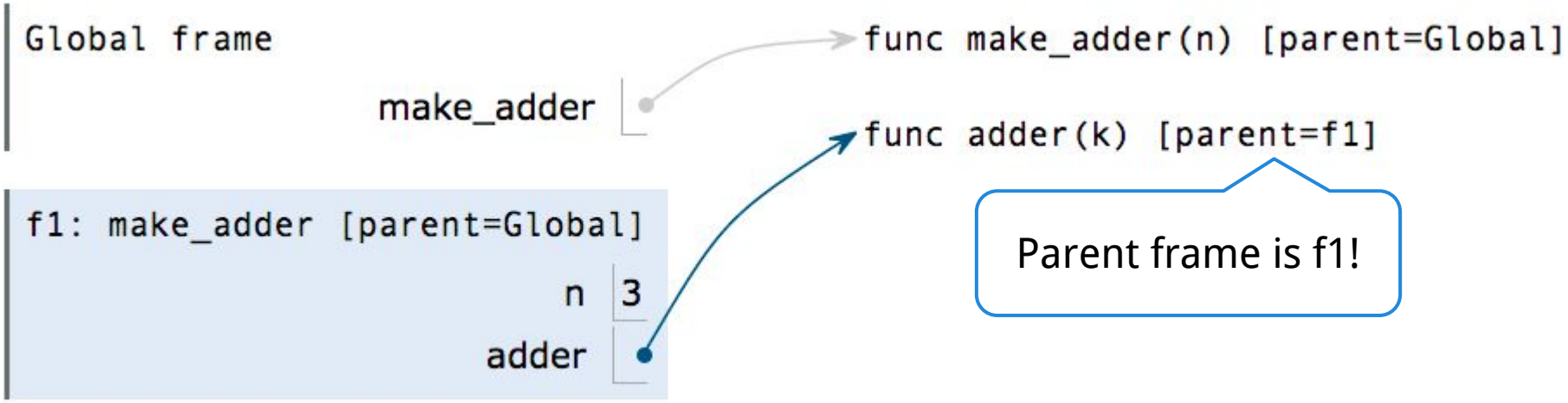
```
def make_adder(n):
    def adder(k):
        return k + n
    return adder
```

```
make_adder(3)(4)
```



Environment diagram: make_adder

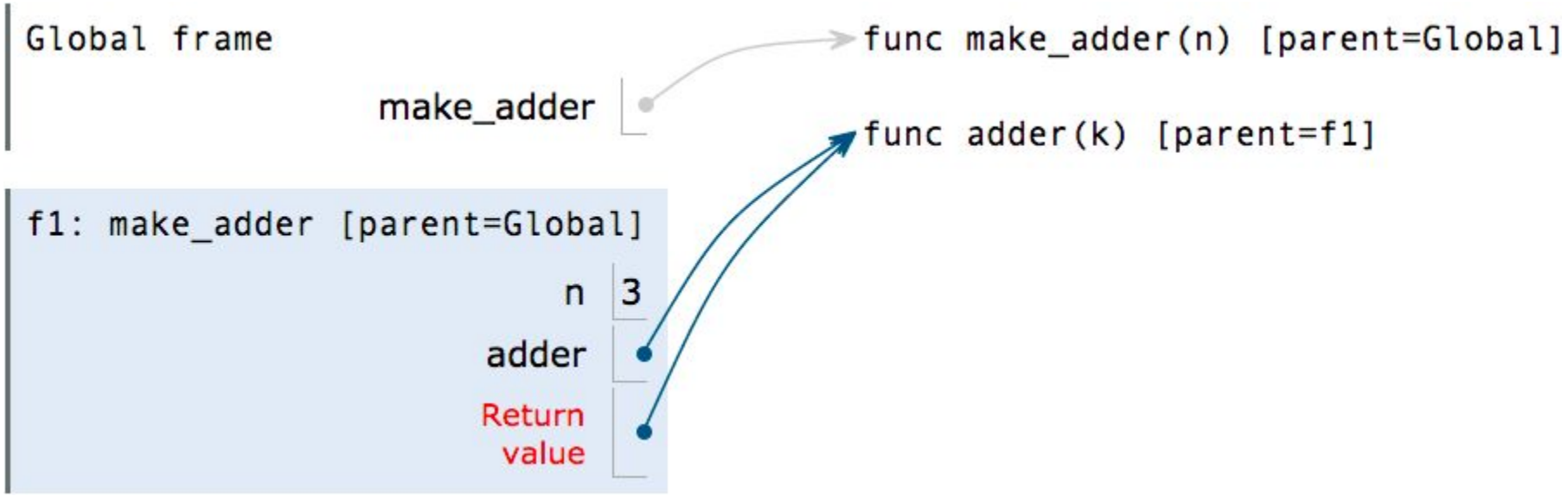
```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

make_adder(3)(4)
```



Environment diagram: make_adder

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

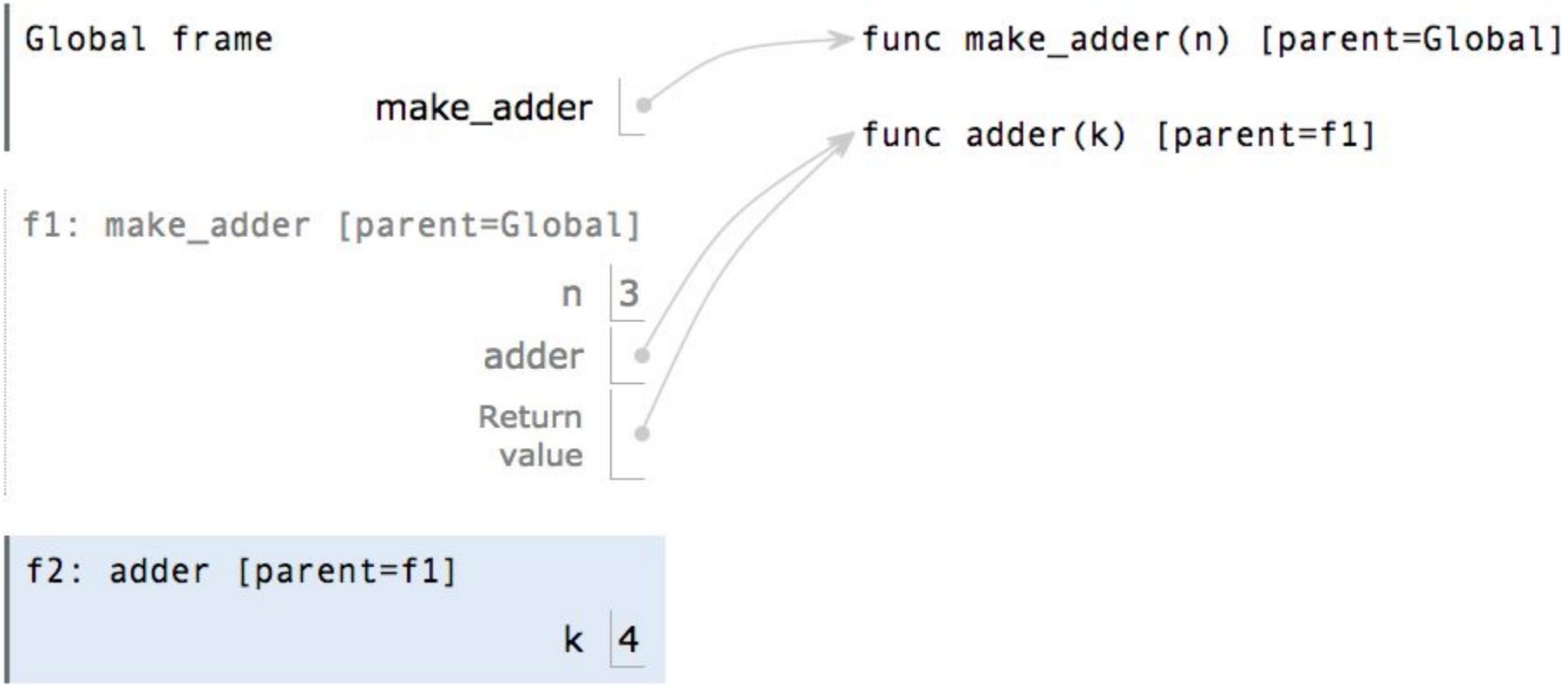
```
make_adder(3)(4)
```



Environment diagram: make_adder

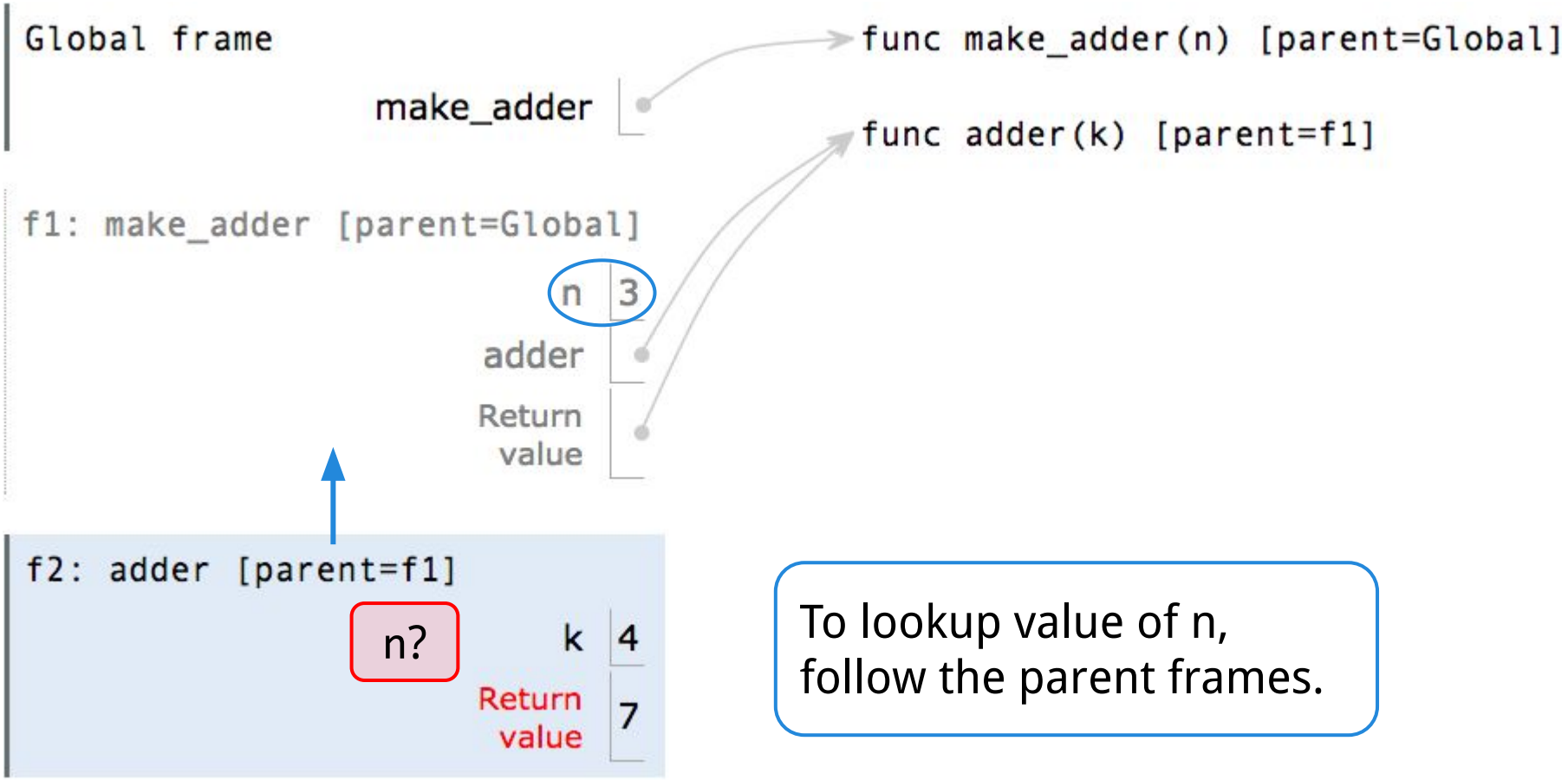
```
def make_adder(n):
    def adder(k):
        return k + n
    return adder
```

▶ `make_adder(3)(4)`



Environment diagram: make_adder

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

make_adder(3)(4)
```



To lookup value of n, follow the parent frames.

Higher Order Functions

1. Review: functions as arguments
2. Nested functions
3. Functions as return values
4. Lambda functions

make_adder

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

Why bother naming `adder` if we only use the name once?

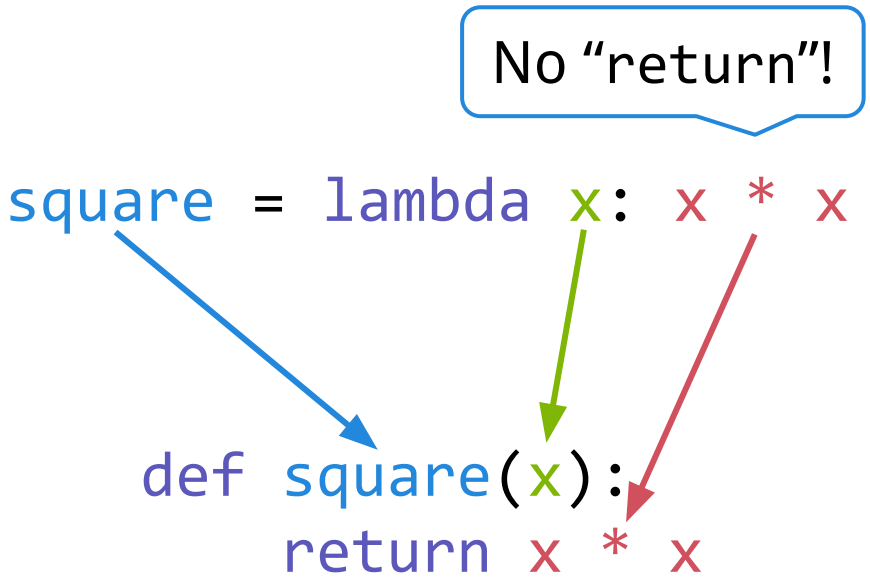
Lambda functions

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

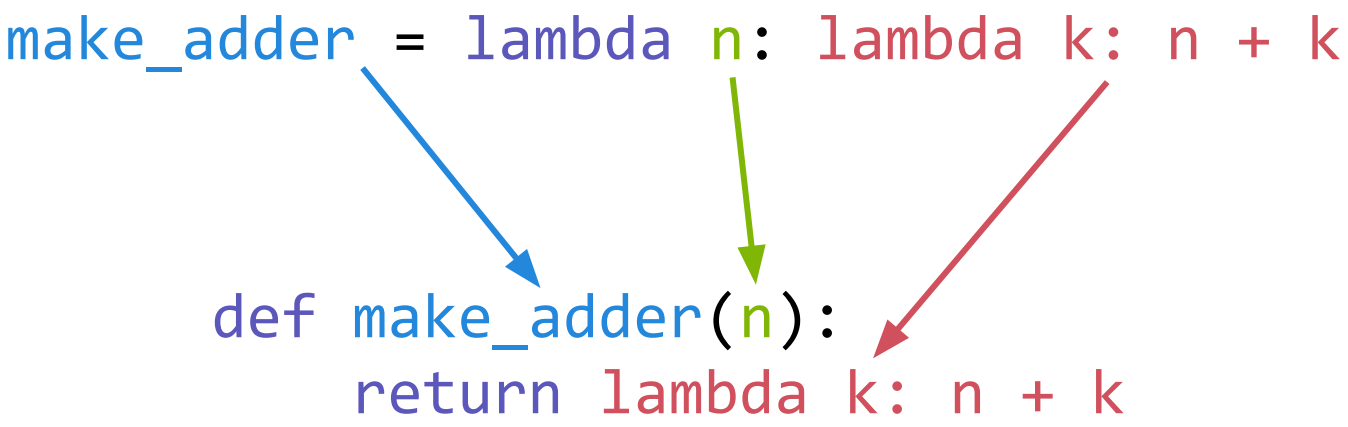
The lambda expression creates an anonymous function!

```
lambda <parameters>: <return expression>
```

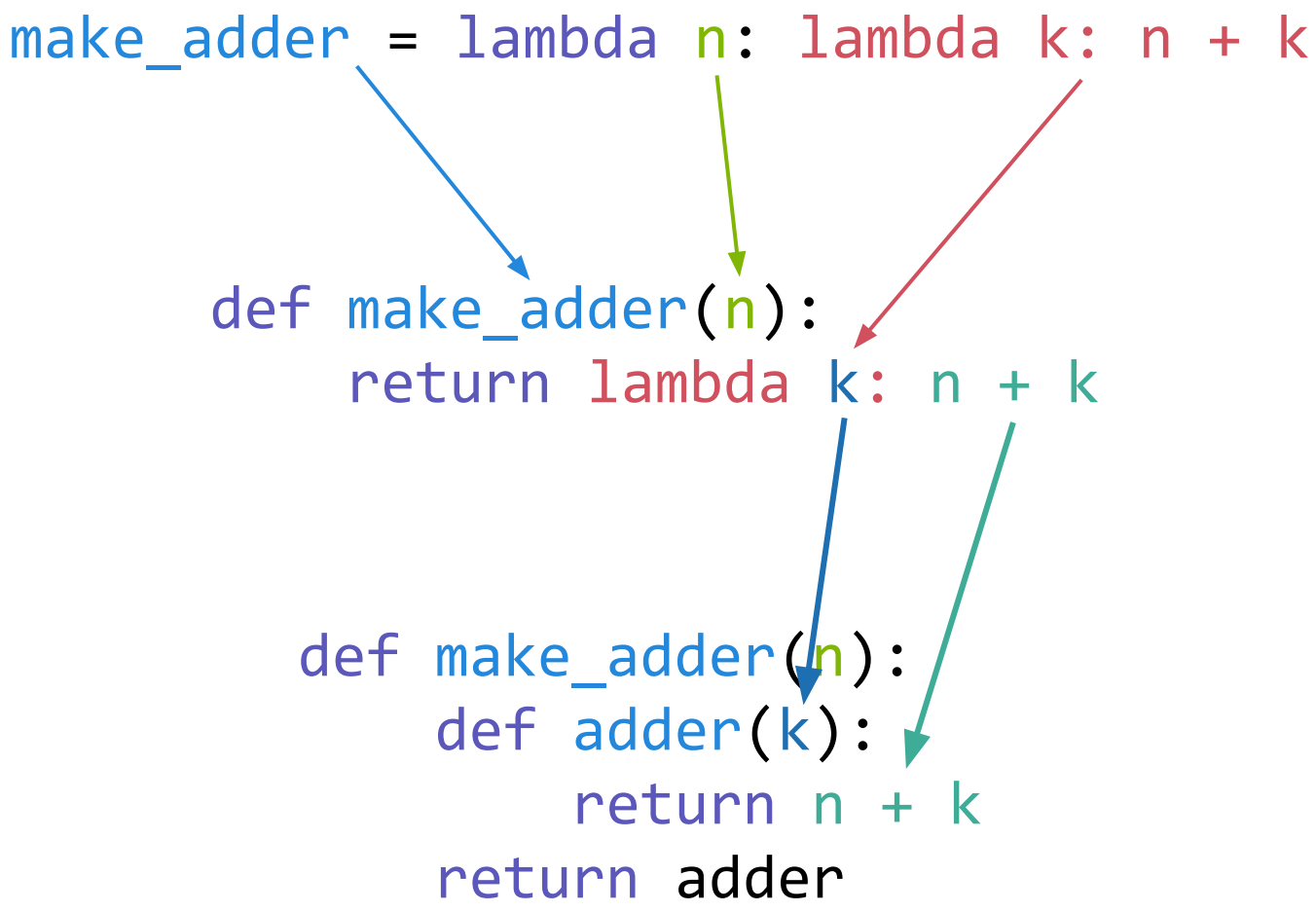
Translating between named and anonymous functions



Translating between named and anonymous functions



Translating between named and anonymous functions



Lambda functions

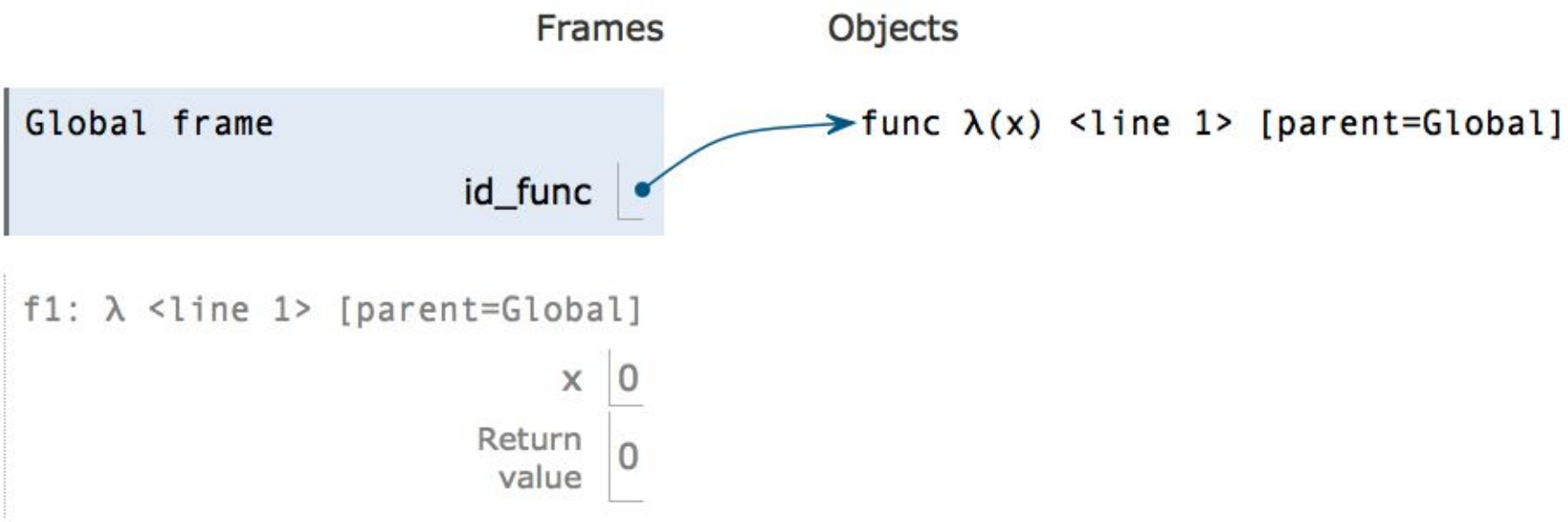
Lambdas are expressions! You can use them anywhere you can use any other expression.

A lambda's body can only be a *single* expression. This is the return expression.

Use sparingly! Better to have clear names.

Environment diagram: lambdas

```
id_func = lambda x: x
id_func(0)
```



Environment diagram: `make_adder`

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

add_3 = make_adder(3)
add_3(4)
```

(Interactive environment diagram)

Lambda functions: practice

```
>>> def summation(term, n):
...     i, total = 1, 0
...     while i <= n:
...         total += term(i)
...         i += 1
...     return total
...
>>> summation(lambda x: x**2, 3)
14
```

Lambda functions: practice

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

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

result = repeat(g, 5)
```

Hog demo!