

# Environments

---

# Announcements

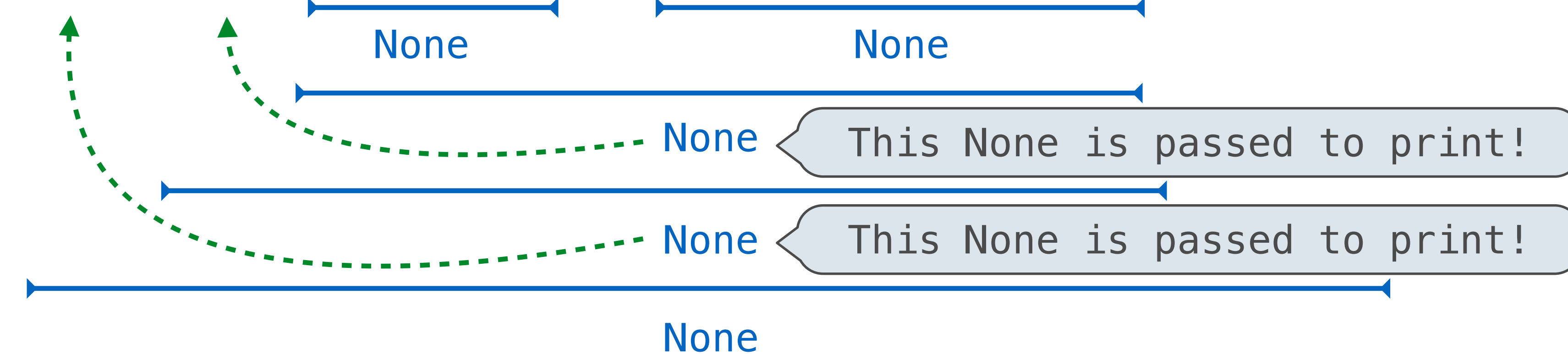
Print and None Review

## Fall 2022 CS 61A Midterm 1, Question 1 (c)

What does the long expression print?

```
s = "Knock"
```

```
print(print(print(s, s) or print("Who's There?")), "Who?")
```



Knock Knock

Who's There?

None

None Who?

False values in Python: `False`, `0`, `'`, `None` (*more to come*)

To evaluate the expression **<left>** `or` **<right>**:

1. Evaluate the subexpression **<left>**.
2. If the result is a true value **v**, then the expression evaluates to **v**.
3. Otherwise, the expression evaluates to the value of the subexpression **<right>**.

# Iteration Review

## Fall 2022 CS 61A Midterm 1, Question 3 It's Perfect

---

A **perfect** number is a positive integer  $n$  whose **proper factors** (the factors of  $n$  below  $n$ ) sum to exactly  $n$ . A number  $n$  is **abundant** if the sum of  $n$ 's proper factors is greater than  $n$  and **deficient** if that sum is less than  $n$ .

Implement `classify`, a function that takes an integer  $n$  greater than 1. It returns the string 'deficient', 'perfect', or 'abundant' that correctly describes  $n$ .

```
>>> classify(6) # Proper factors 1, 2 and 3 sum to exactly 6.
'perfect'
>>> classify(24) # Proper factors 1, 2, 3, 4, 6, 8, and 12 sum to 36.
'abundant'
>>> classify(23) # Proper factor 1 sums to 1.
'deficient'
```

[pollev.com/cs61a](https://pollev.com/cs61a)

What do we need to compute?

- (a) print the factors less than  $n$
- (b) count the factors less than  $n$
- (c) sum the factors less than  $n$
- (d) find the biggest factor less than  $n$

What should we iterate over?

- (a) the factors less than  $n$
- (b) every integer less than  $n$
- (c) every integer less than or equal to  $n$

## Fall 2022 CS 61A Midterm 1, Question 3 (a) It's Perfect

---

```
def classify(n):
    """Return whether n > 1 is 'deficient', 'perfect', or 'abundant'.
    >>> classify(6) # Proper factors 1, 2 and 3 sum to exactly 6.
    'perfect'
    >>> classify(24) # Proper factors 1, 2, 3, 4, 6, 8, and 12 sum to 36.
    'abundant'
    >>> classify(23) # Proper factor 1 sums to 1.
    'deficient'
    """
    total, k = 0, 1
    while _____:
        if _____:
            _____
            k = k + 1
    if total == n:
        return 'perfect'
    elif _____:
        return 'deficient'
    else:
        return 'abundant'
```

**What are we doing?**

Sum the factors below n

Iterate over every integer below n, check if it's a factor

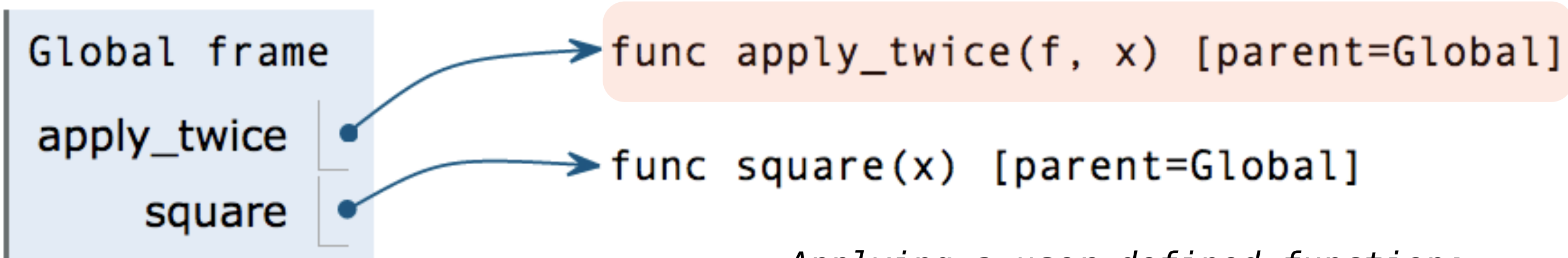
## Environments for Higher-Order Functions

Student advice from the Fall 2024 final survey:

"ENVIRONMENT DIAGRAMS ARE EXTREMELY IMPORTANT! Taking this class with no prior Python experience and minimal overall programming experience, taking time to understand environment diagrams helped me fully understand step-by-step how my code is interpreted, and any areas where my code may be going wrong. This made coding more intuitive for me, as it helped me gain a understanding of the connections being made between my code and carried out functions."

# Names can be Bound to Functional Arguments

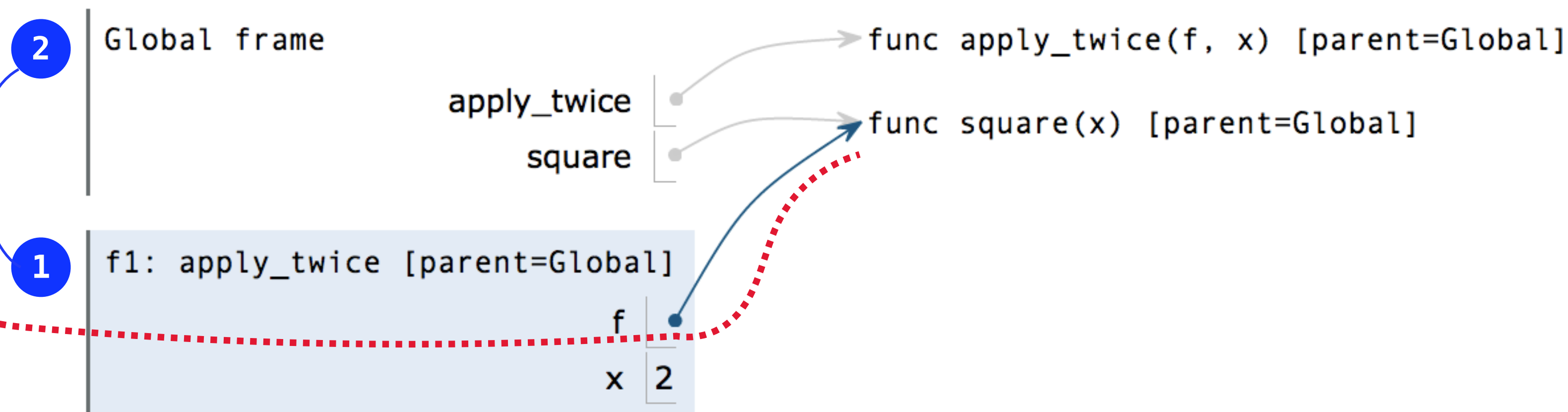
```
1 def apply_twice(f, x):  
2     return f(f(x))  
3  
→ 4 def square(x):  
5     return x * x  
6  
→ 7 result = apply_twice(square, 2)
```



*Applying a user-defined function:*

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

```
→ 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)
```



# Environment Diagrams for Nested Def Statements

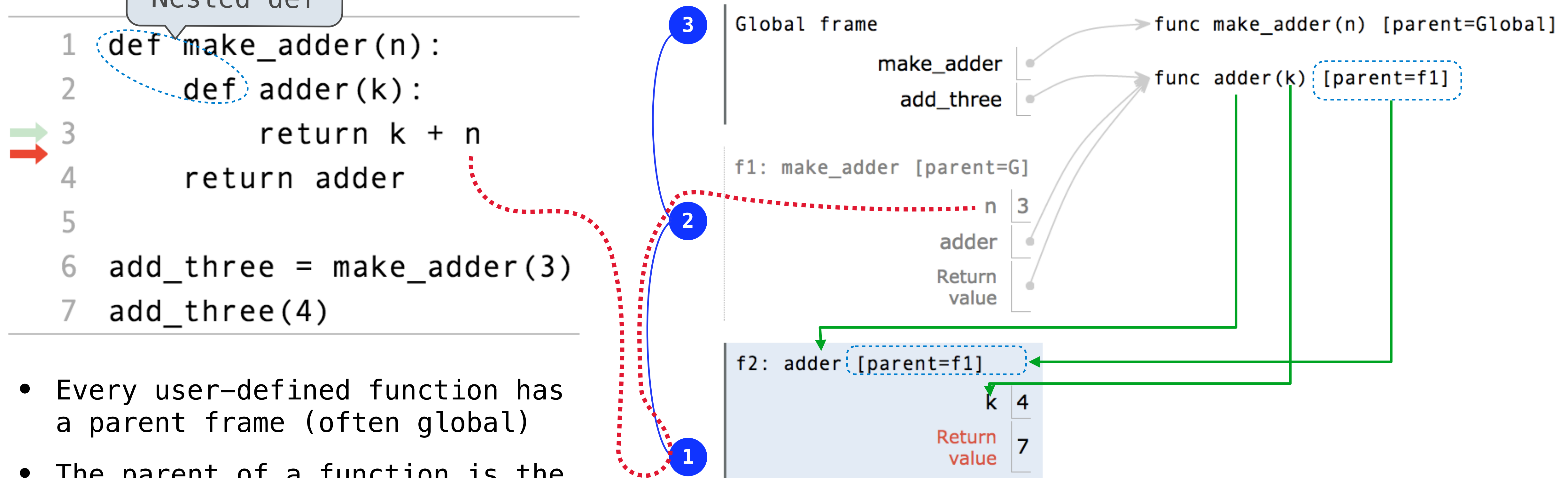
## Nested def

```

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)

```

- 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



## Wrong make\_adder()?

```
def make_adder(n):  
    k = 5  
    def adder(k):  
  
        return n + k  
  
    return adder
```

(A)

```
def make_adder(n):  
    n = 5  
    def adder(k):  
  
        return n + k  
  
    return adder
```

(B)

```
def make_adder(n):  
  
    def adder(k):  
        k = 5  
        return n + k  
  
    return adder
```

(C)

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

(D)

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

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

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

```
add_three = make_adder(3)  
add_five = make_adder(5)  
print(add_three(4))
```

Which of these implementations **do not print 7**?

[pollev.com/cs61a](https://pollev.com/cs61a)

# How to Draw an Environment Diagram

When a function is defined:

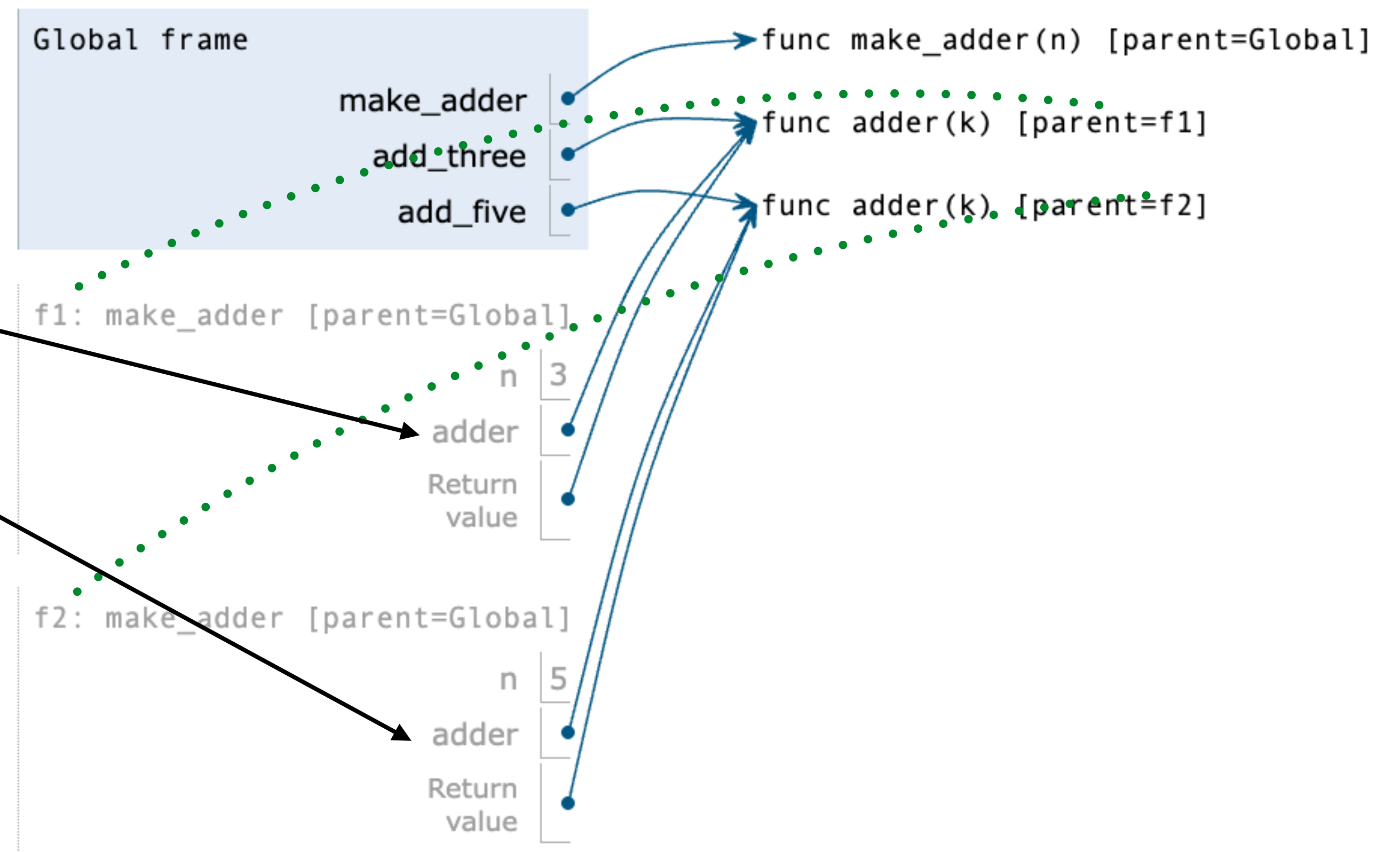
Create a function value: `func <name>(<formal parameters>) [parent=<label>]`

Its parent is the current frame.

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

When a function is called:

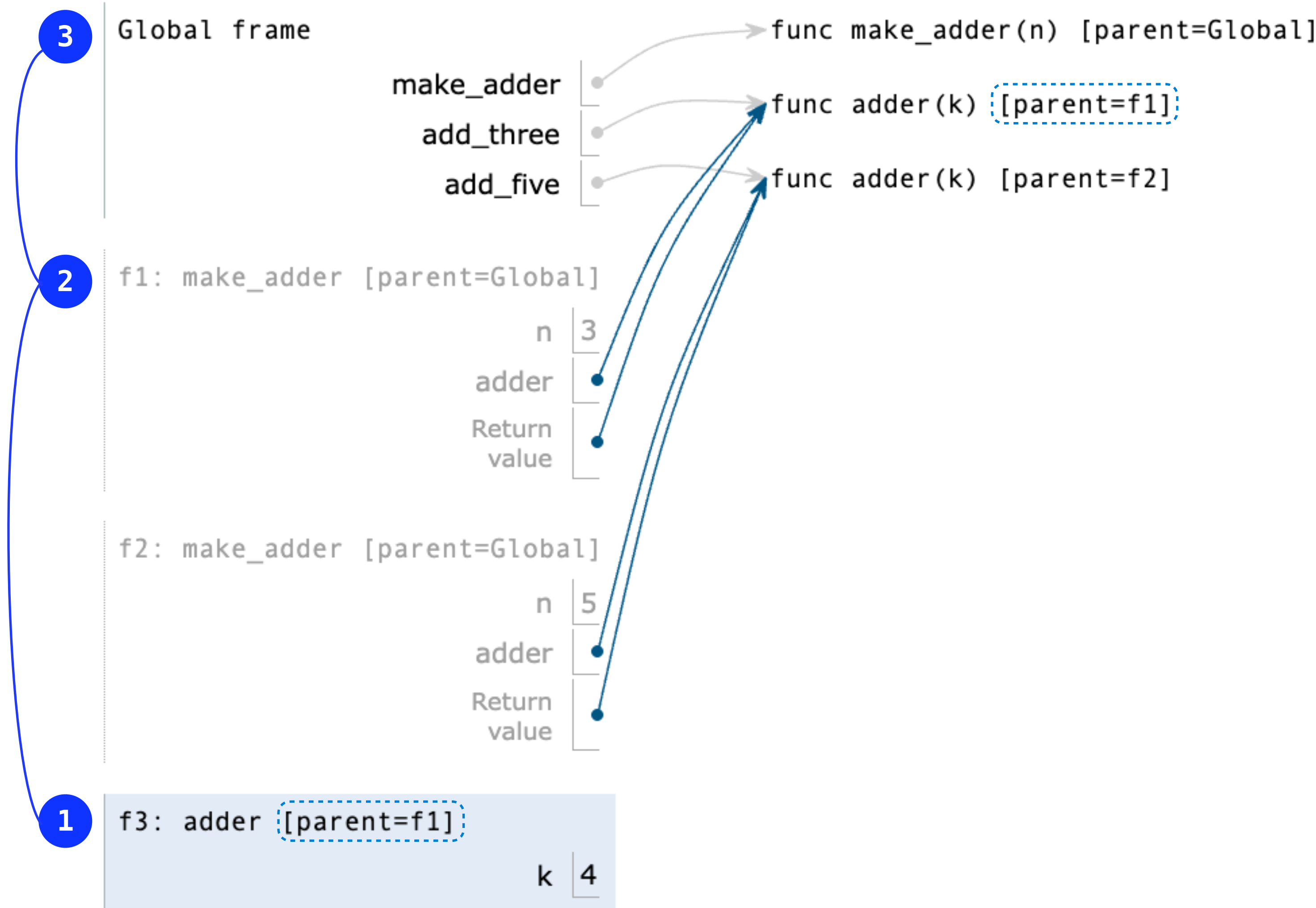
1. Add a local frame, titled with the <name> of the function being called.
- ★ 2. Copy the parent of the function to the local frame: `[parent=<label>]`
3. Bind the <formal parameters> to the arguments in the local frame.



# How to Draw an Environment Diagram

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.



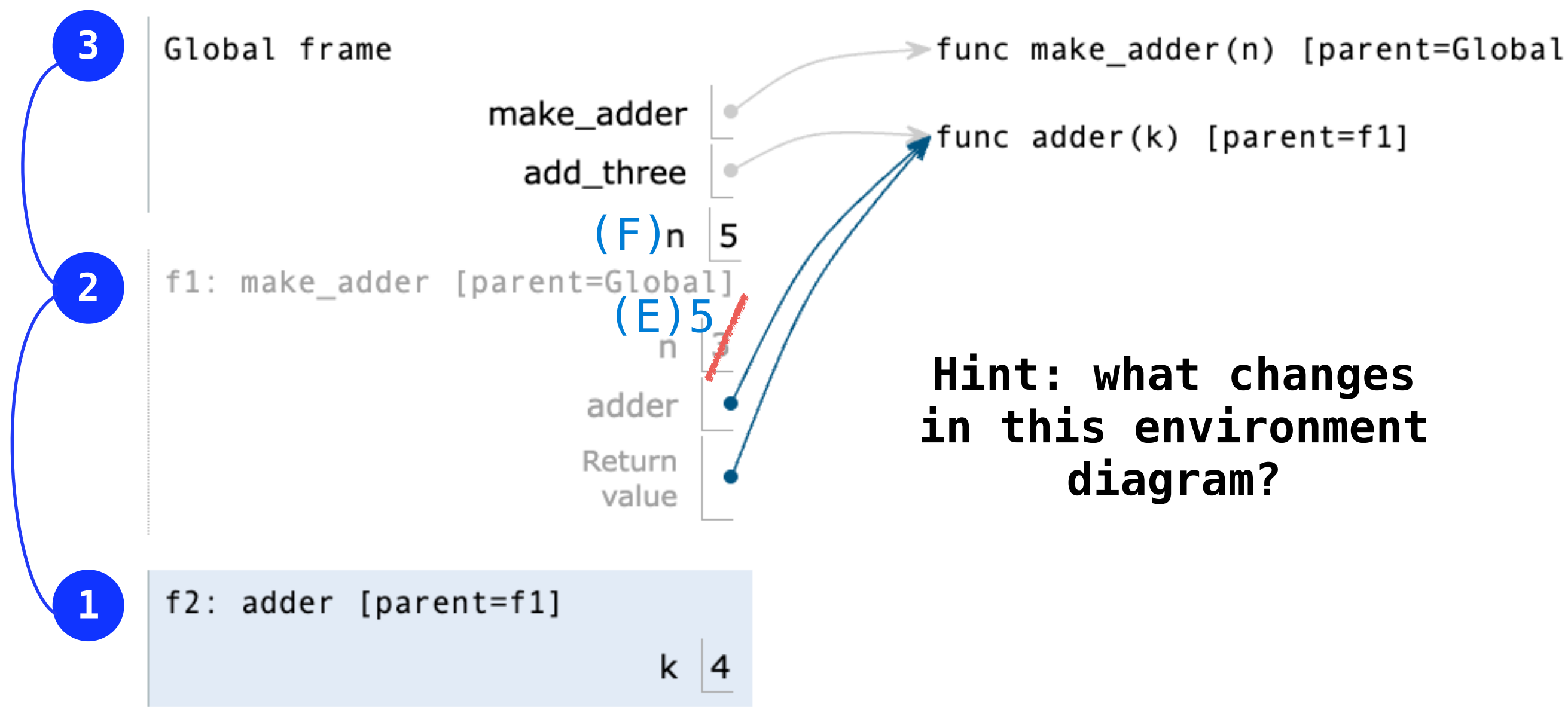
# Wrong make\_adder()?

```
def make_adder(n):  
    def adder(k):  
        return n + k  
    n = 5  
    return adder  
  
add_three = make_adder(3)  
print(add_three(4))
```

(E)

```
def make_adder(n):  
    def adder(k):  
        return n + k  
    return adder  
  
add_three = make_adder(3)  
n = 5  
print(add_three(4))
```

(F)



Which of these implementations results in an incorrect add\_three function, so prints a **different result**?

[pollev.com/cs61a](https://pollev.com/cs61a)

## Wrong make\_adder()?

---

---

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

---

**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