

CS 134 Lecture 13:

Scope

Announcements & Logistics

- **Lab 04 Feedback is out!** Can you interpret TestResults.txt?
- Lab 03 Graded feedback is out
- **HW 5** will due tonight @ 10pm
- **Lab 4 Part 2** due Wednesday/Thursday 10pm
- **Midterm reminders:**
 - **Review: Monday 3/11** from 7-9pm
 - **Exam Thurs 3/14** from 6-7:30pm OR 8-9:30pm
 - Both exam and review are in Bronfman Auditorium

Do You Have Any Questions?

Last Time: Aliasing

- Attempts to change **immutable** objects (e.g., strings) produce **clones**
 - Changes to clones do not affect originals
 - No aliasing!
- We can create **aliases** of **mutable** objects
 - Aliases refer to the same object, so changes to that object through any alias affect value that other aliases observe
- For the list data type, **+=** is sneakily replaced by **.append()**
 - This mutates the list!

Goal was to demystify surprising behavior:
nothing in computer science is magic!

Today's Plan

- **Scope:** variables, functions, objects have limited accessibility/visibility.
- Understanding how this works helps us make decisions about where to define variables/functions/objects

Goal is to again demystify surprising behavior:
nothing in computer science is magic!

Review: Scope Example from Lecture 4

```
def my_func (val):  
    val = val + 1  
    print('local val', val)  
    return val
```

```
val = 3  
new_val = my_func(val)  
print('global val', val)
```

What is printed here?

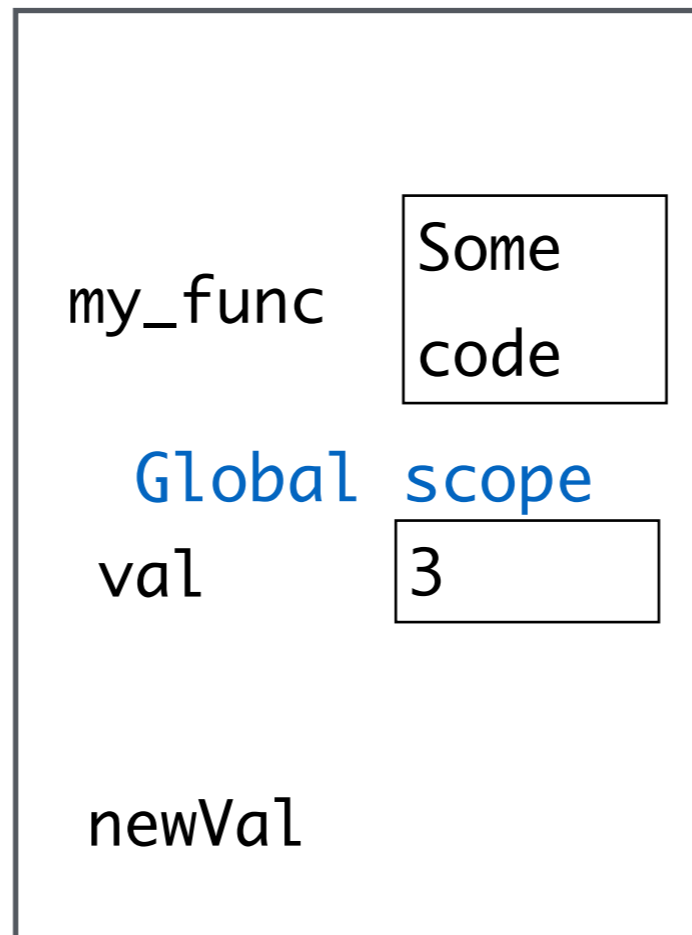
What is returned?

What is printed here?

Review: Scope Example from Lecture 4

```
def my_func (val):  
    val = val + 1  
    print('local val', val)  
    return val
```

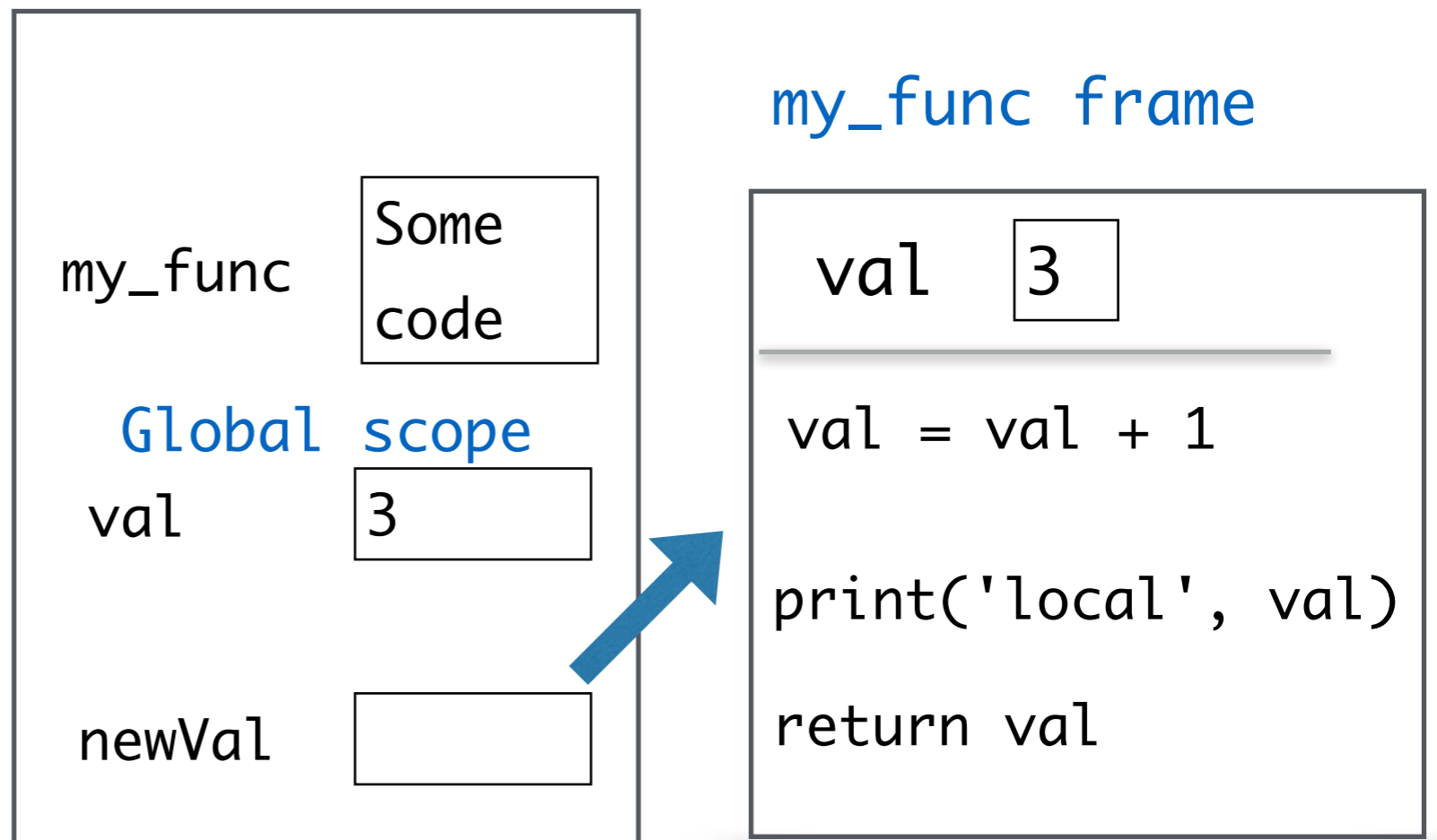
```
val = 3  
new_val = my_func(val)  
print('global val', val)
```



Review: Scope Example from Lecture 4

```
def my_func (val):  
    val = val + 1  
    print('local val', val)  
    return val
```

```
val = 3  
new_val = my_func(val)  
print('global val', val)
```

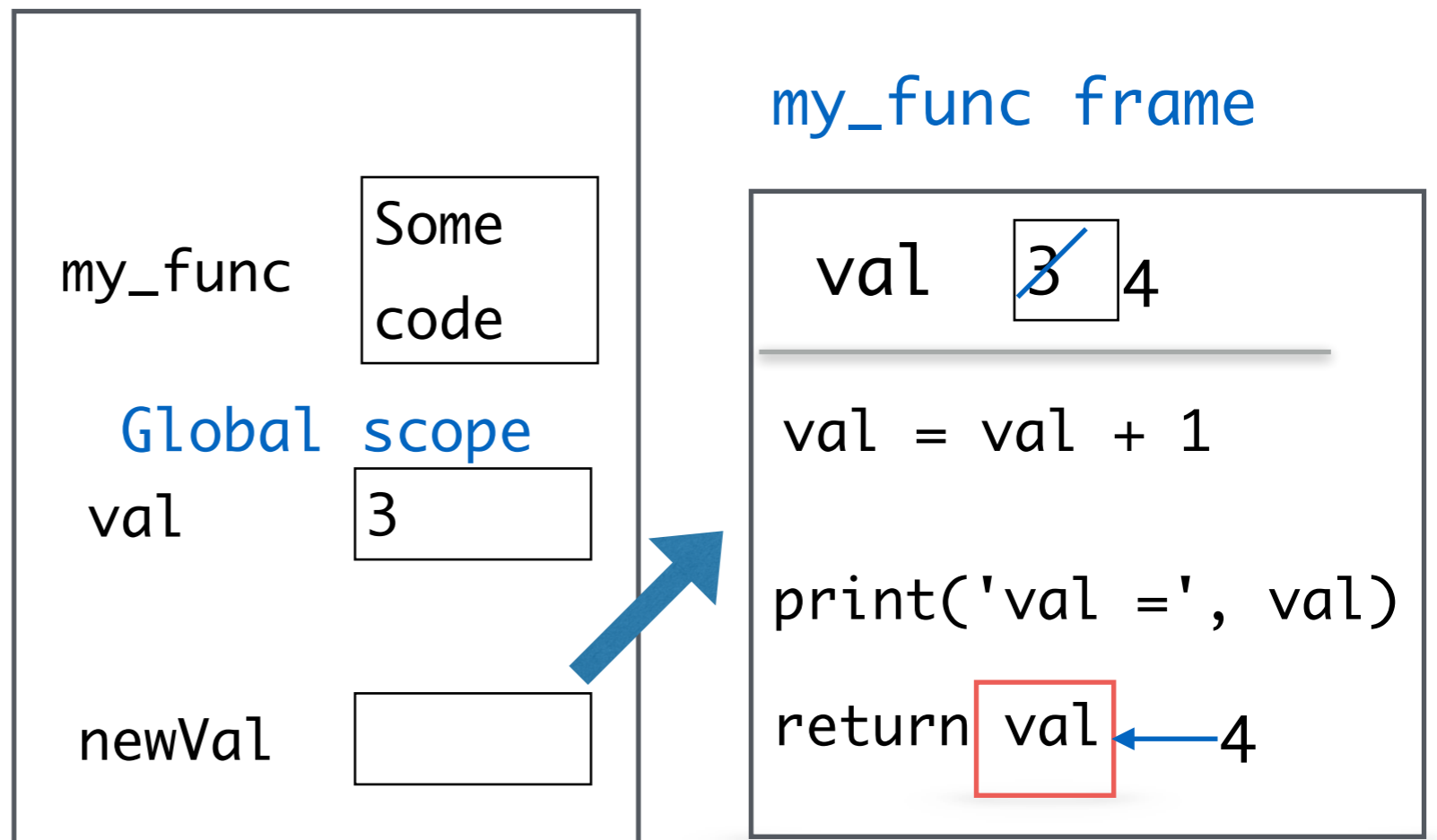


Review: Scope Example from Lecture 4

```
def my_func (val):  
    val = val + 1  
    print('local val', val)  
    return val
```

```
val = 3  
new_val = my_func(val)  
print('global val', val)
```

What is printed here?

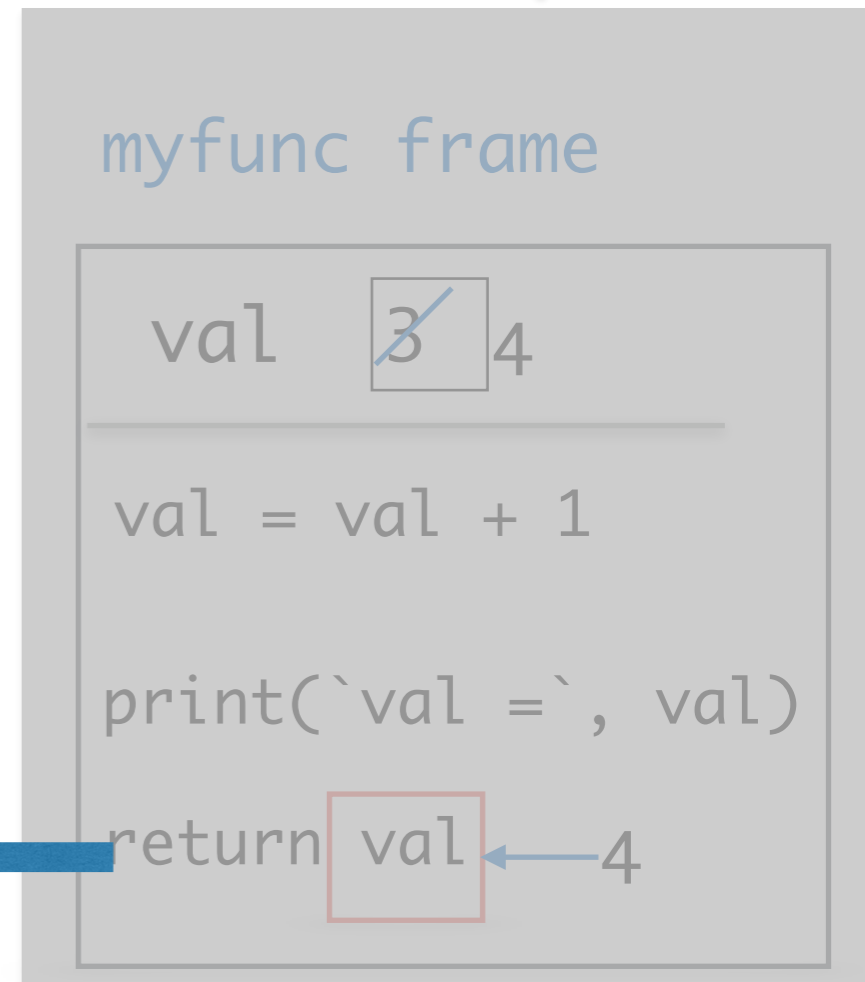
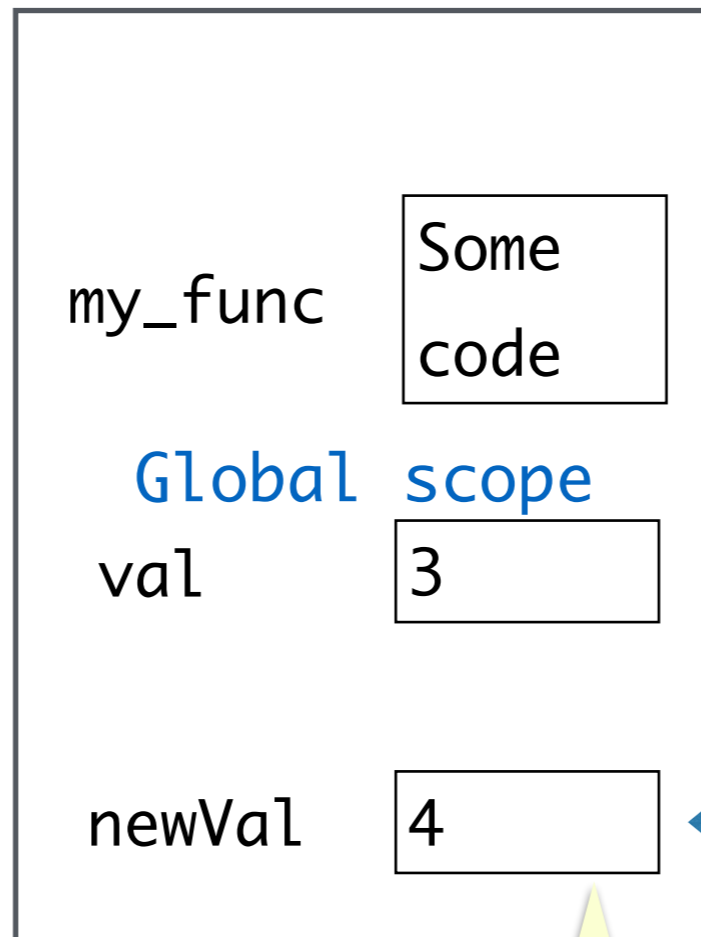


Review: Scope Example from Lecture 4

```
def my_func (val):  
    val = val + 1  
    print('local val', val)  
    return val
```

```
val = 3  
new_val = my_func(val)  
print('global val', val)
```

Function frame destroyed
(and all local variables lost)
after return from call



Information flow out of a function is only through return statements!

What gets printed to the screen?

```
a = 3
```

```
b = 4
```

```
def square(x):  
    return x * x
```

?

```
sum_sq = square(a) + square(b)
```

```
c = sum_sq ** 0.5
```

```
print(c)
```

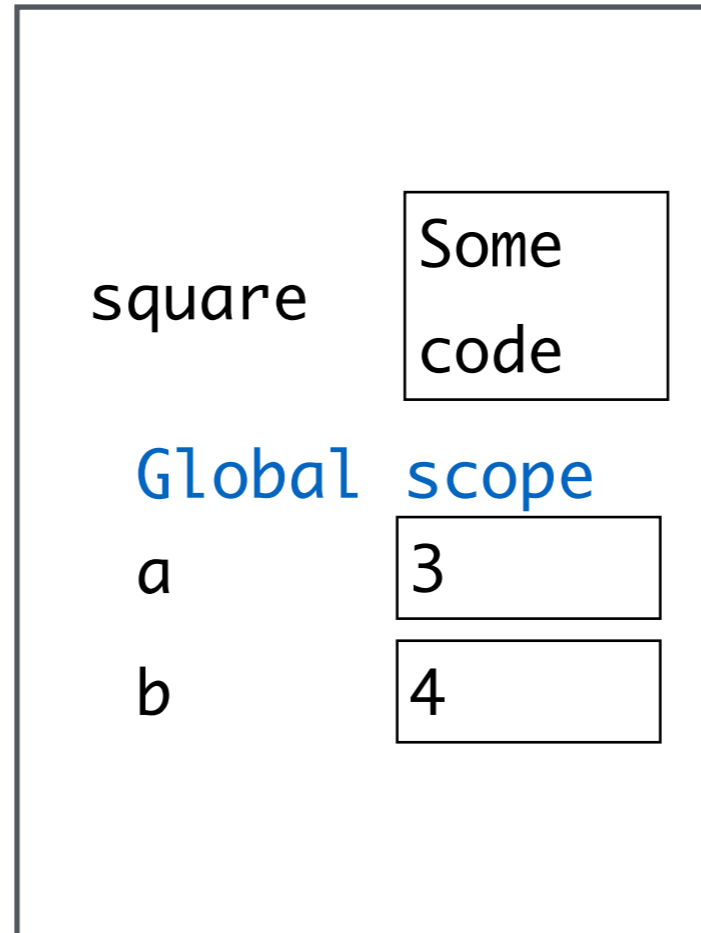
What is printed here?

Understanding Scope

```
a = 3
```

```
b = 4
```

```
def square(x):  
    return x * x
```

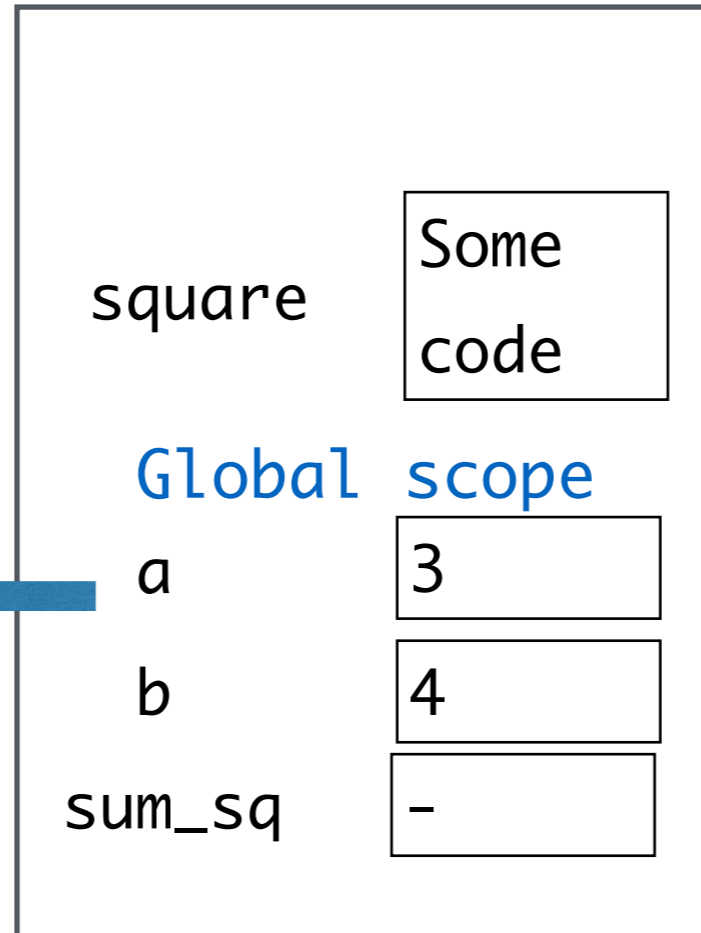


Understanding Scope

```
a = 3  
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = square(a) +  
        square(b)
```

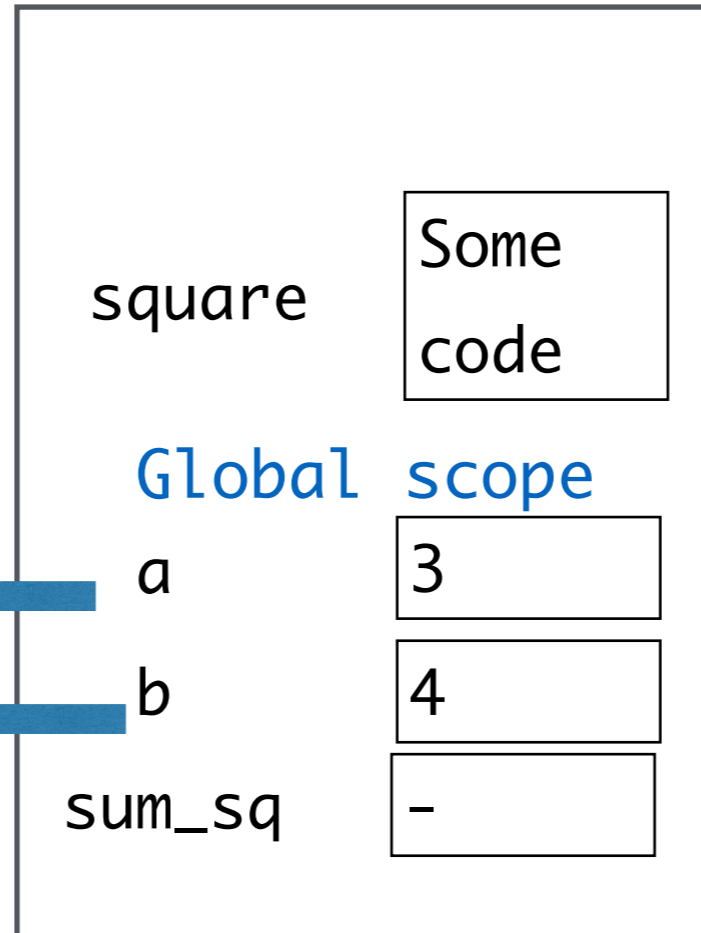


Understanding Scope

```
a = 3  
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = square(3) +  
         square(b) ?
```

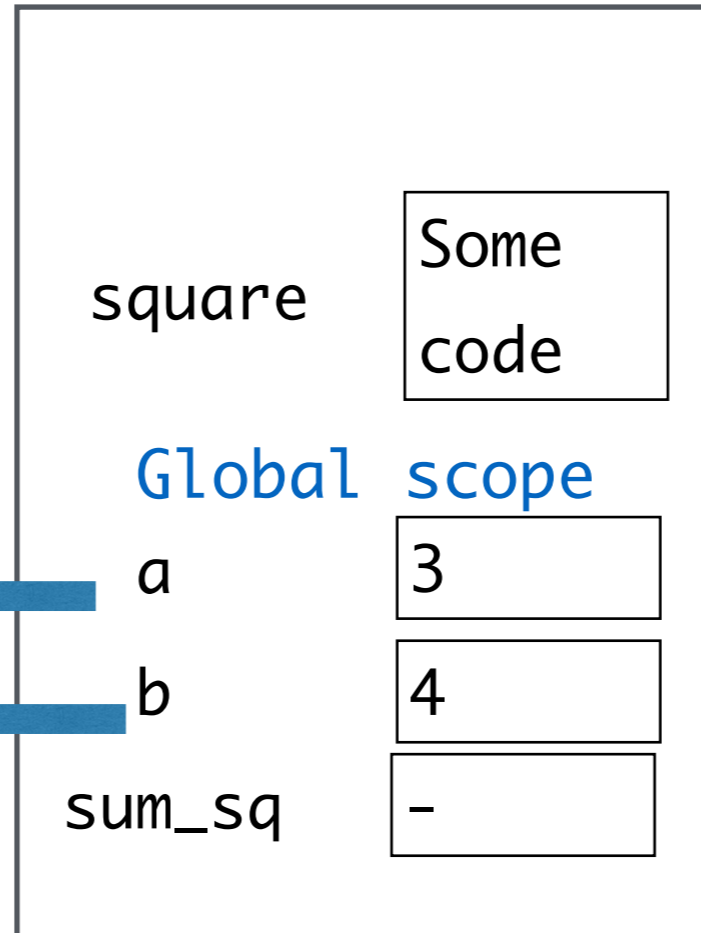


Understanding Scope

```
a = 3  
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = square(3) +  
          square(4) ?
```

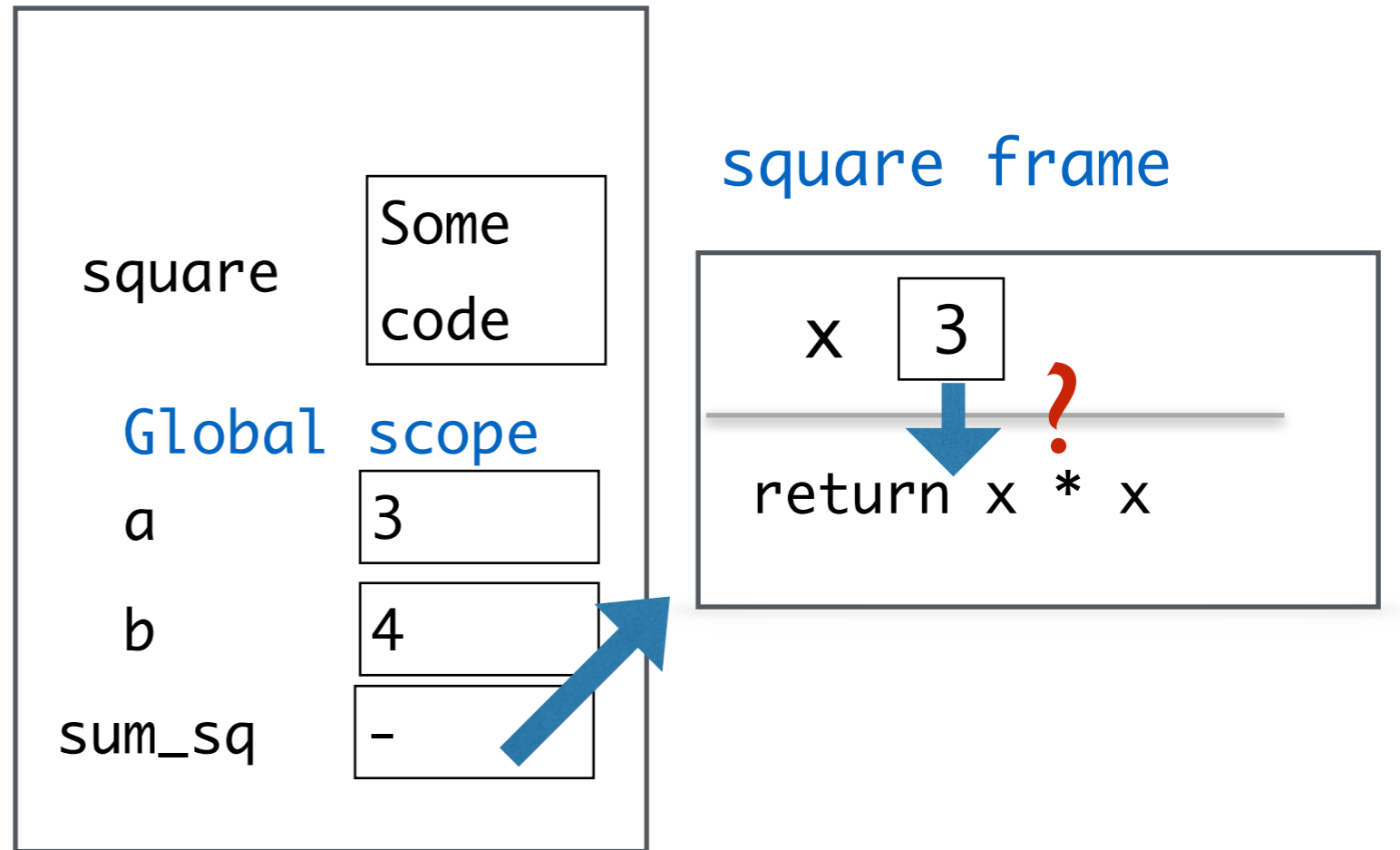


Understanding Scope

```
a = 3  
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = square(3) +  
        square(4)
```



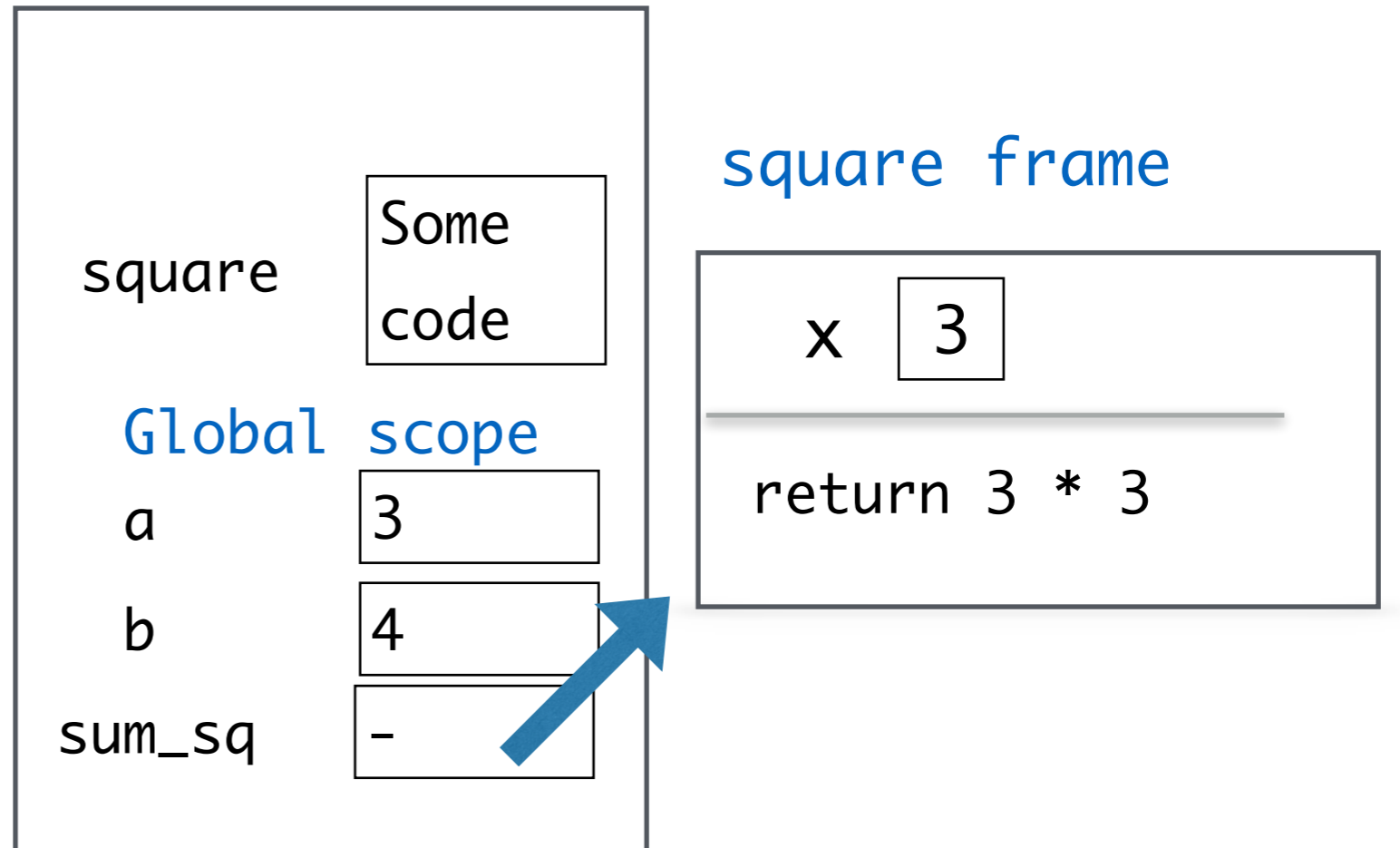
Understanding Scope

```
a = 3
```

```
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = square(3) +  
         square(4)
```



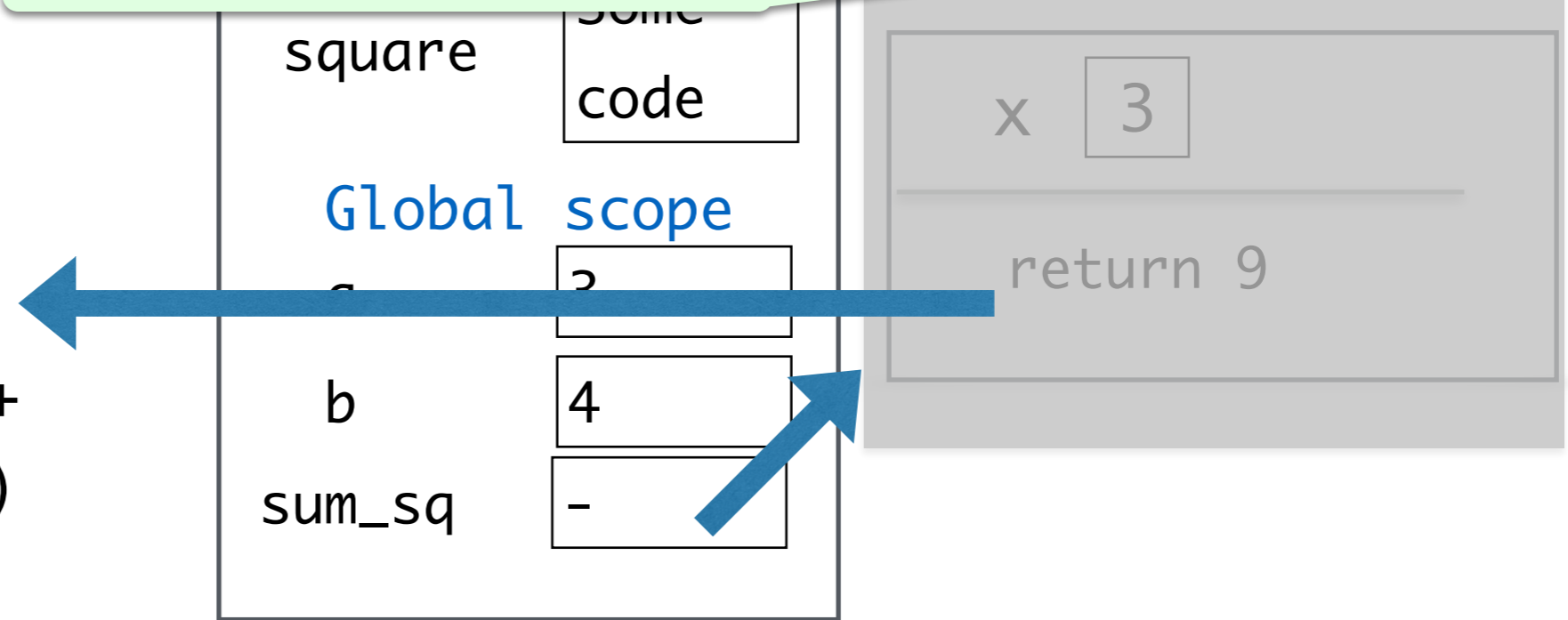
Understanding Scope

a = 3
b = 4

```
def square(x):  
    return x * x
```

```
sum_sq = square(3) +  
         square(4)
```

Function frame destroyed (and all local variables lost) after return from function call

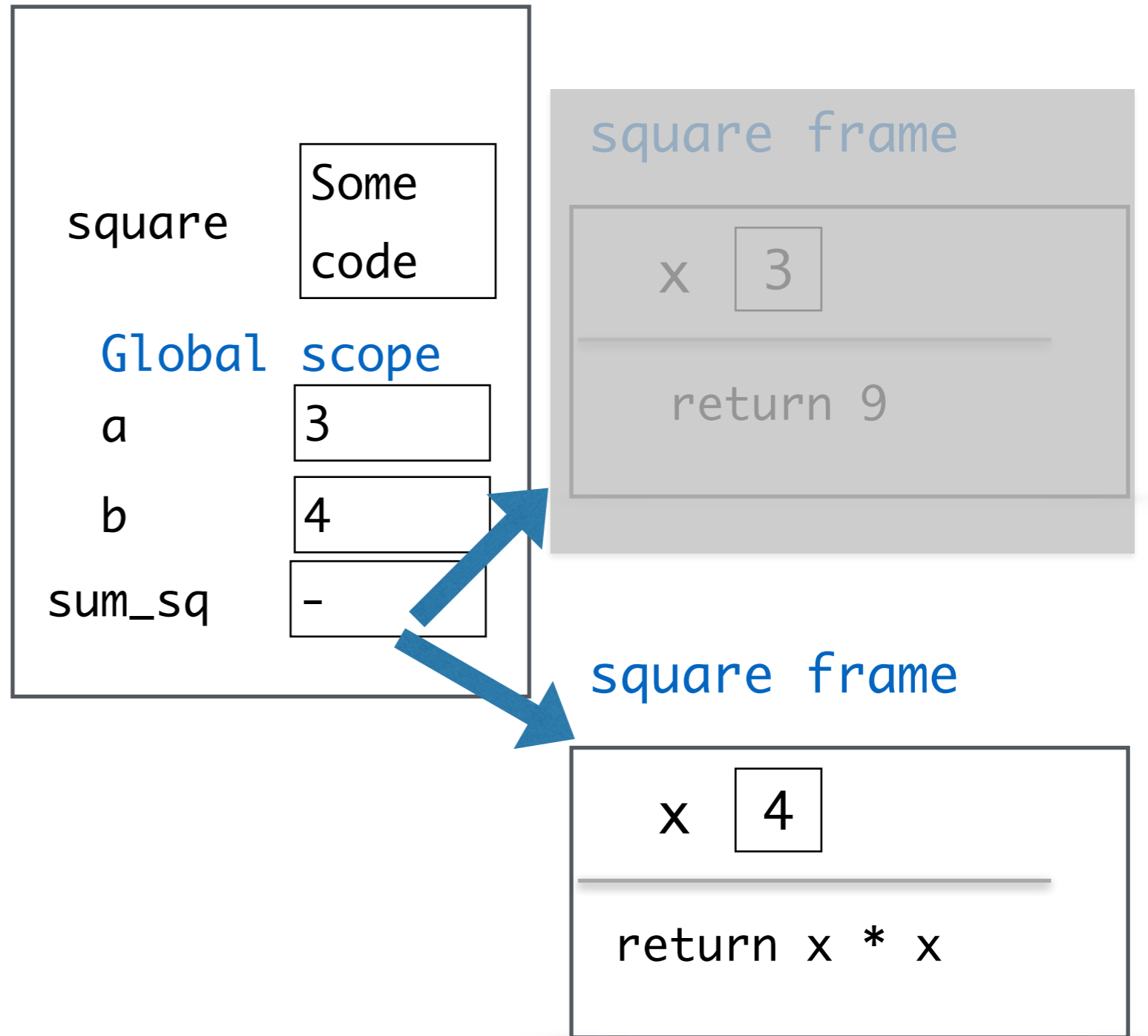


Understanding Scope

```
a = 3  
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = 9 +  
    square(4)
```

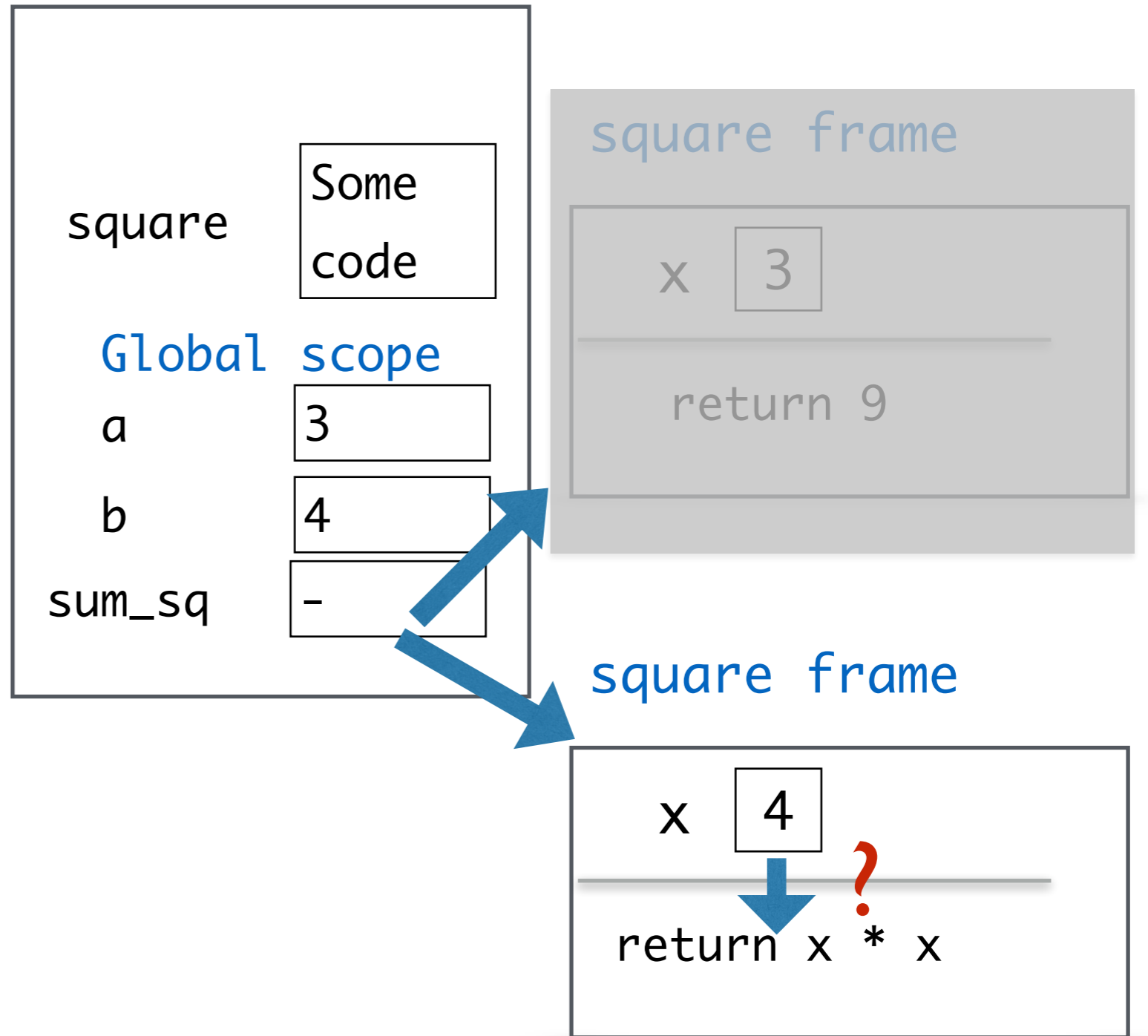


Understanding Scope

```
a = 3  
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = 9 +  
    square(4)
```

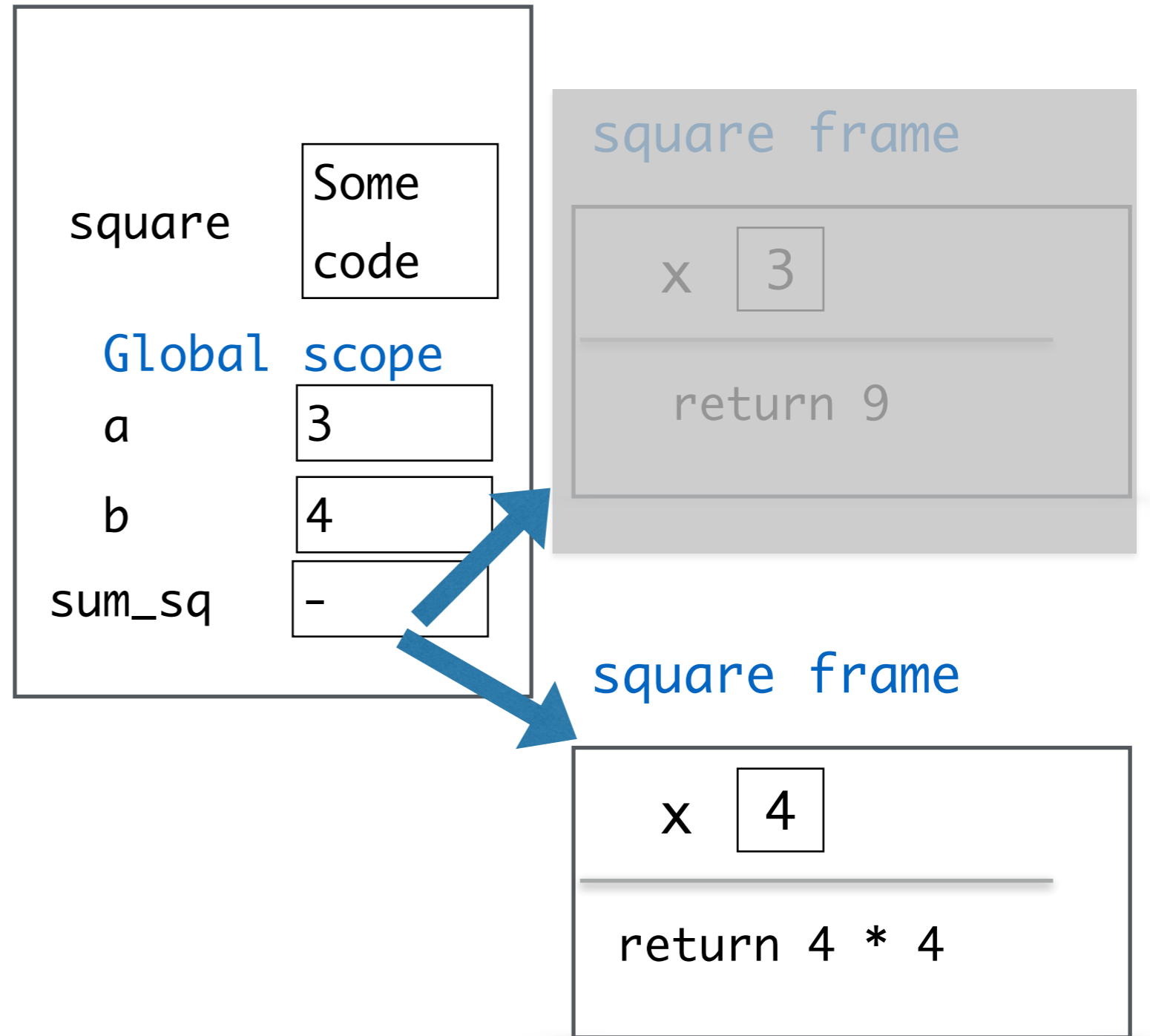


Understanding Scope

```
a = 3  
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = 9 +  
    square(4)
```

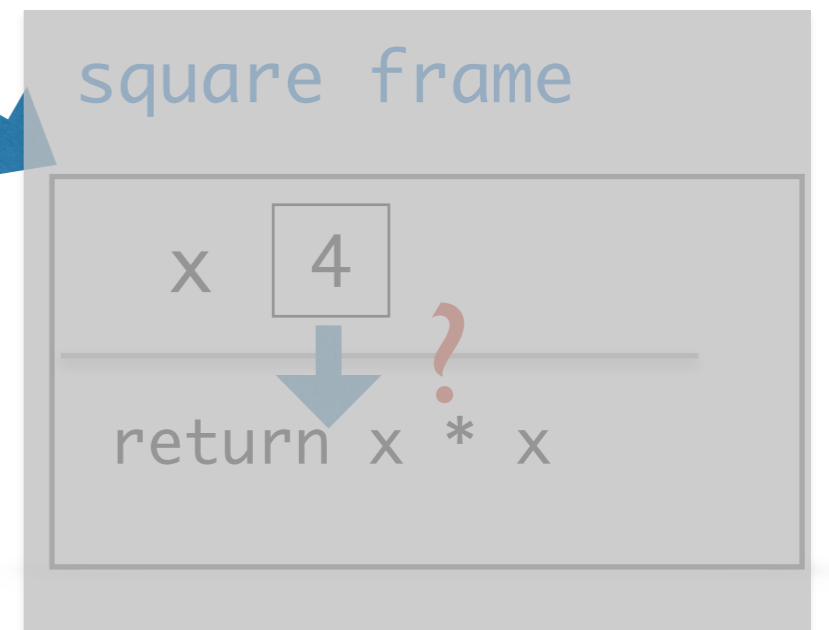
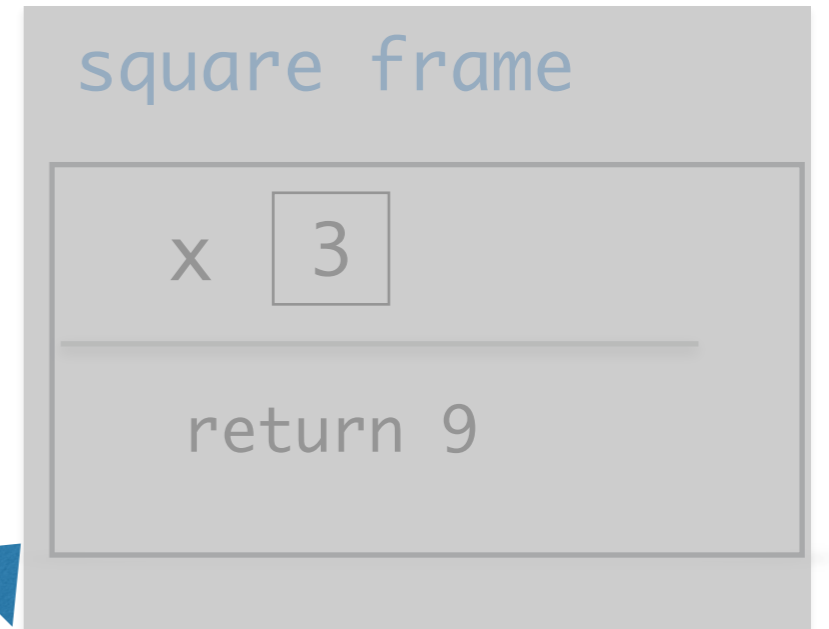
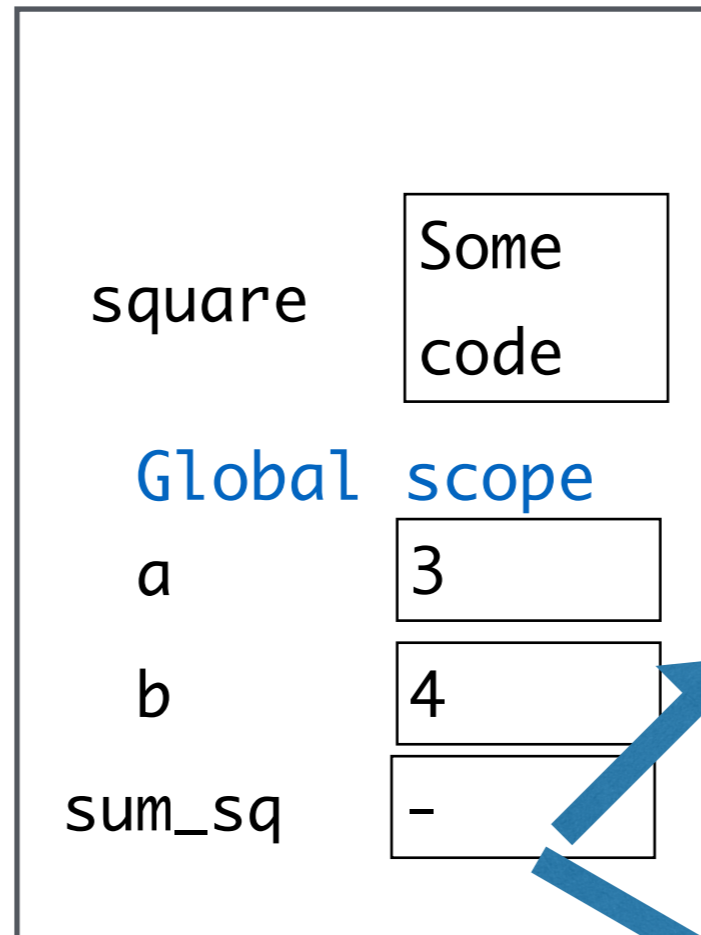


Understanding Scope

```
a = 3  
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = 9 +  
square(4)
```

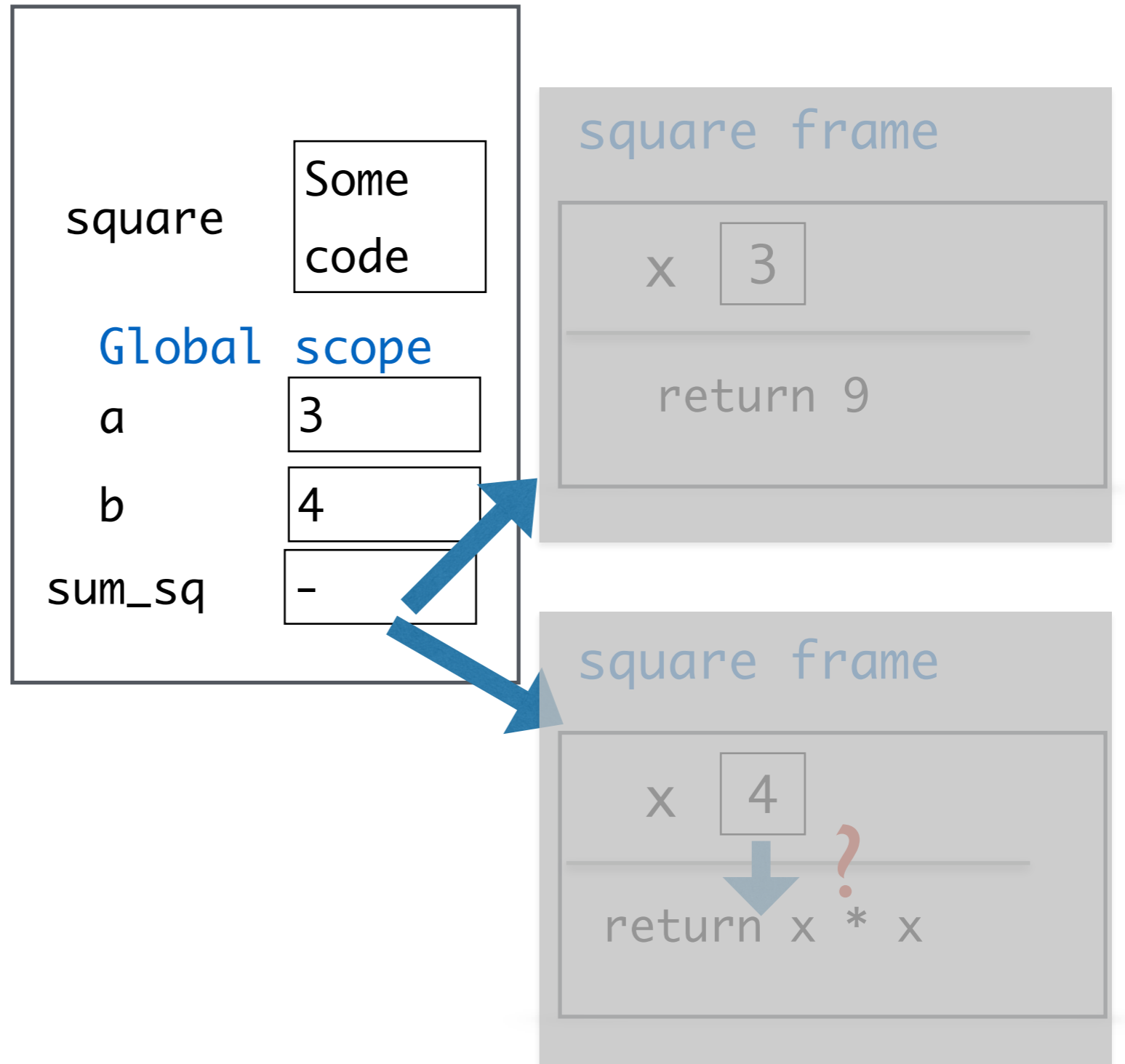


Understanding Scope

```
a = 3  
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = 9 +  
        16
```



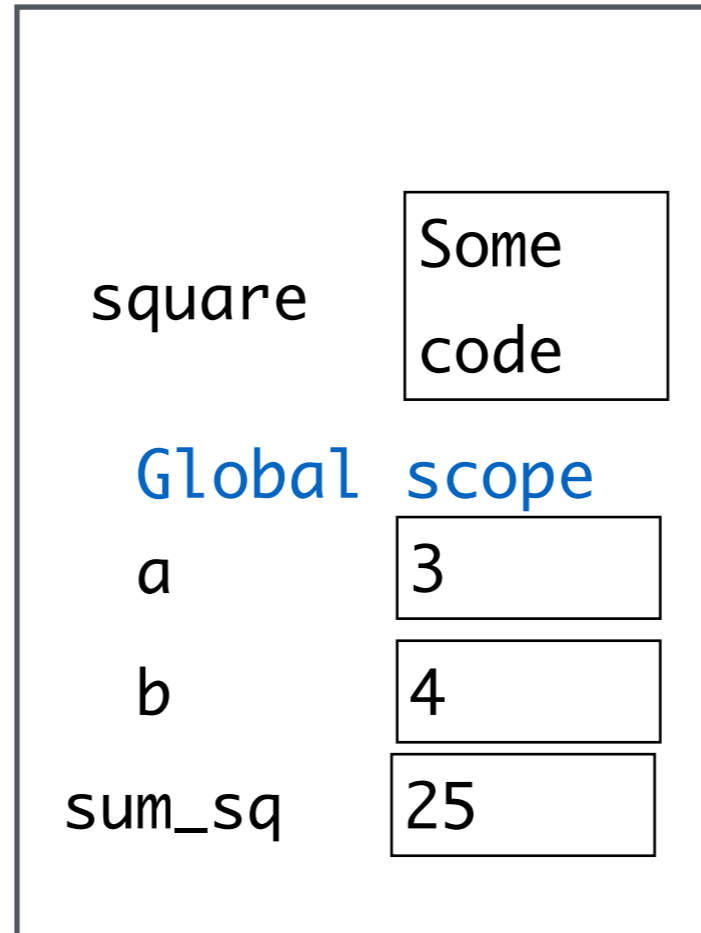
Understanding Scope

```
a = 3
```

```
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = 25
```



Understanding Scope

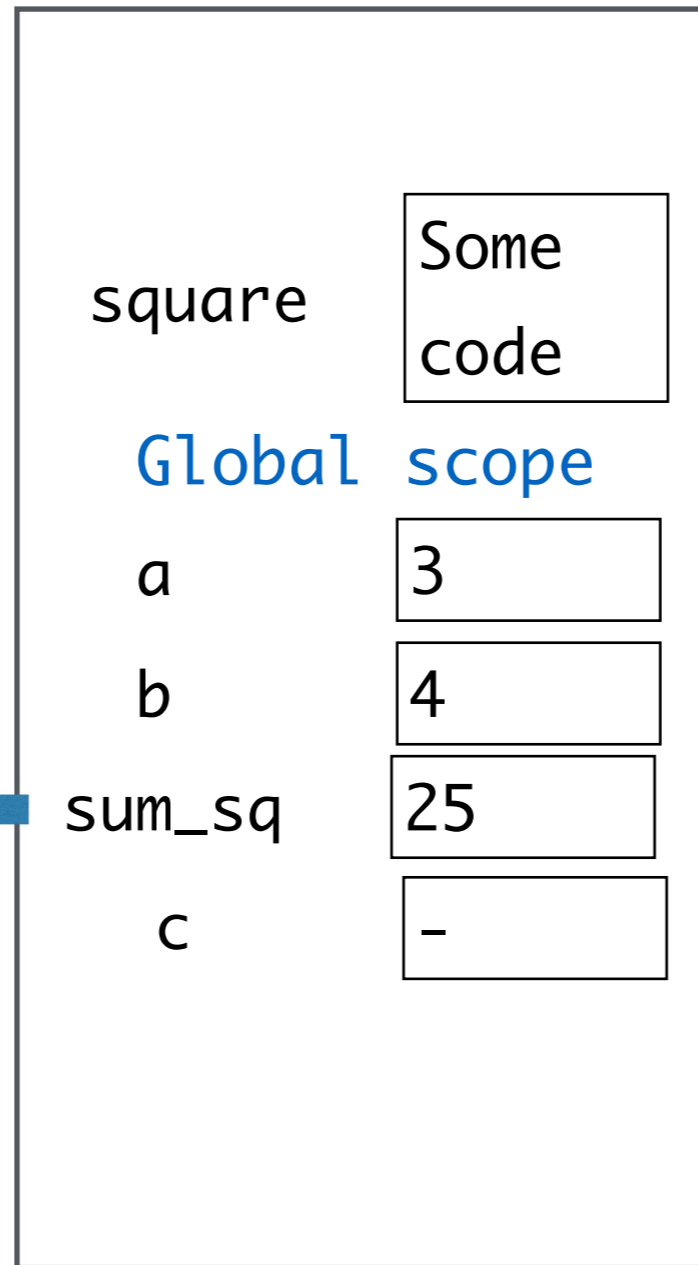
```
a = 3  
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = 25
```

```
c = sum_sq ** 0.5
```

```
print(c)
```



Understanding Scope

```
a = 3
```

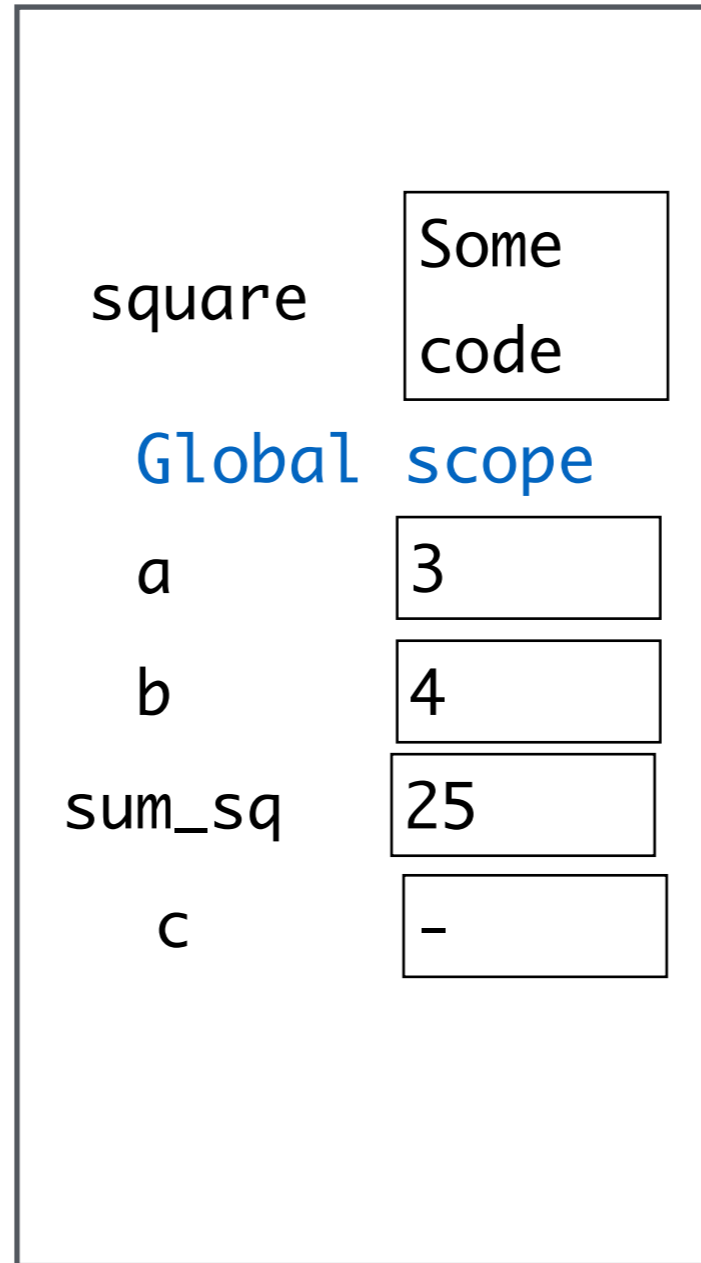
```
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = 25
```

```
c = 25 ** 0.5
```

```
print(c)
```



Understanding Scope

```
a = 3
```

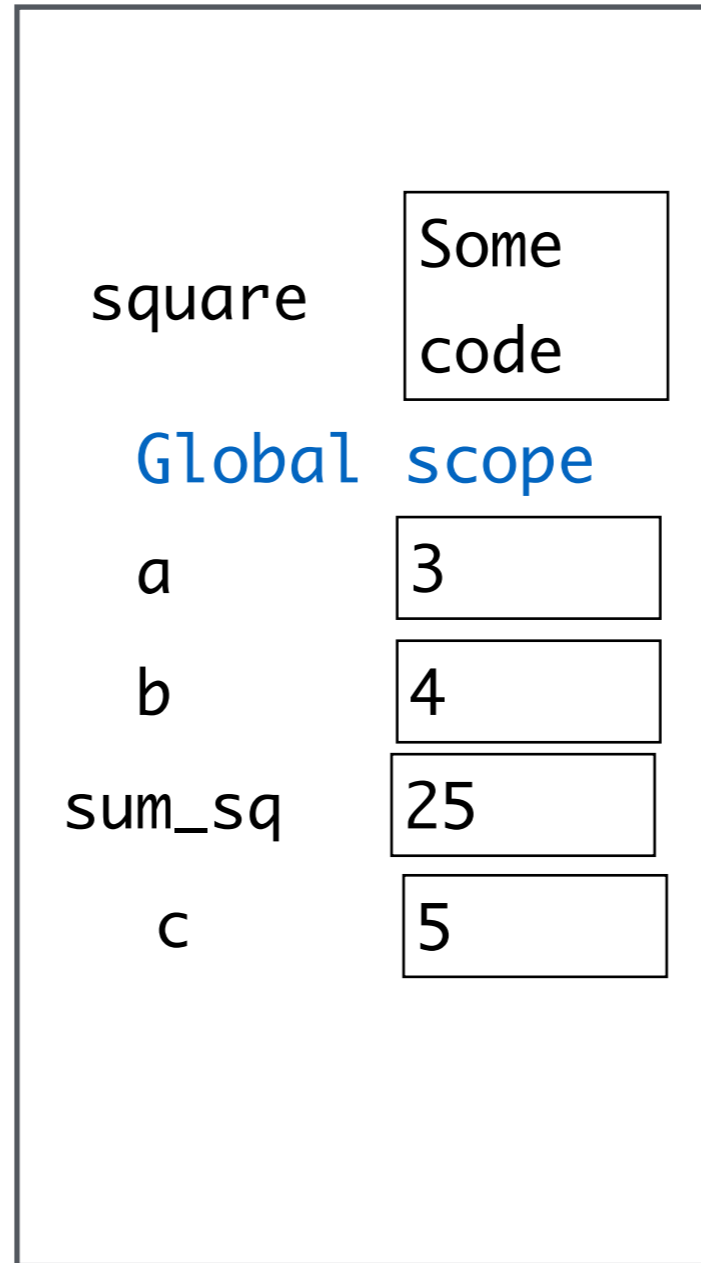
```
b = 4
```

```
def square(x):  
    return x * x
```

```
sum_sq = 25
```

```
c = 5
```

```
print(c)
```



Finally, **5** is printed

Local Parameter Names

```
a = 3
```

```
b = 4
```

What if we change this?

```
def square(a):  
    return a * a
```

```
sum_sq = square(a) + square(b)
```

```
c = sum_sq ** 0.5
```

```
print(c)
```

Does it change the behavior?

Local Parameter Names

```
a = 3
```

```
b = 4
```

```
def square(a):  
    return b * b
```

How about this change?

Will it throw **NameError**?

```
sum_sq = square(a) + square(b)
```

```
c = sum_sq ** 0.5
```

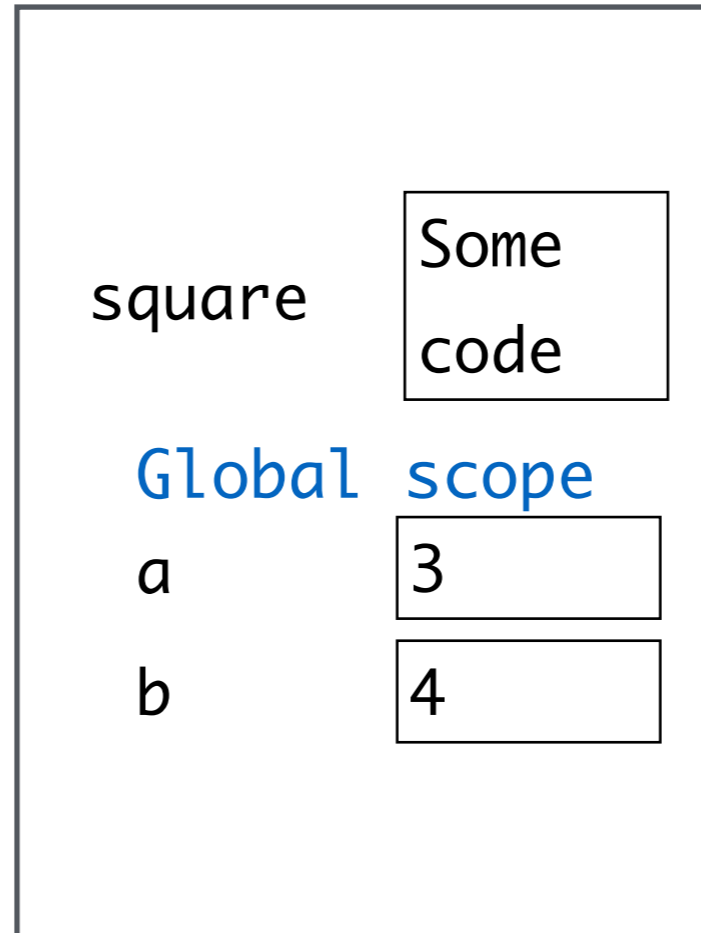
```
print(c)
```

Understanding Scope

```
a = 3
```

```
b = 4
```

```
def square(a):  
    return b * b
```

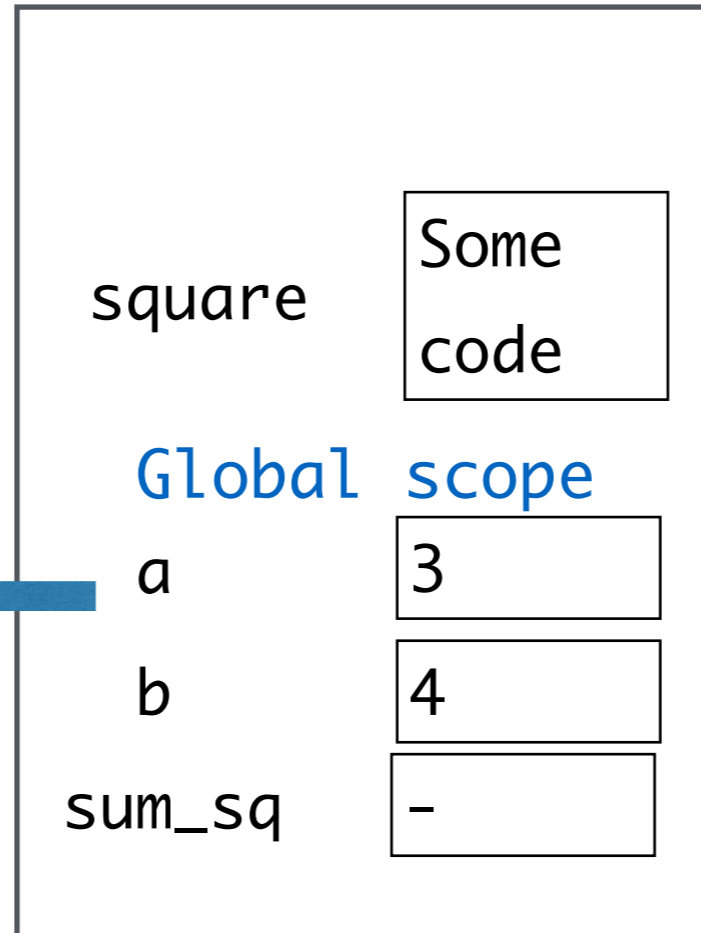


Understanding Scope

```
a = 3  
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = square(a) +  
        square(b)
```

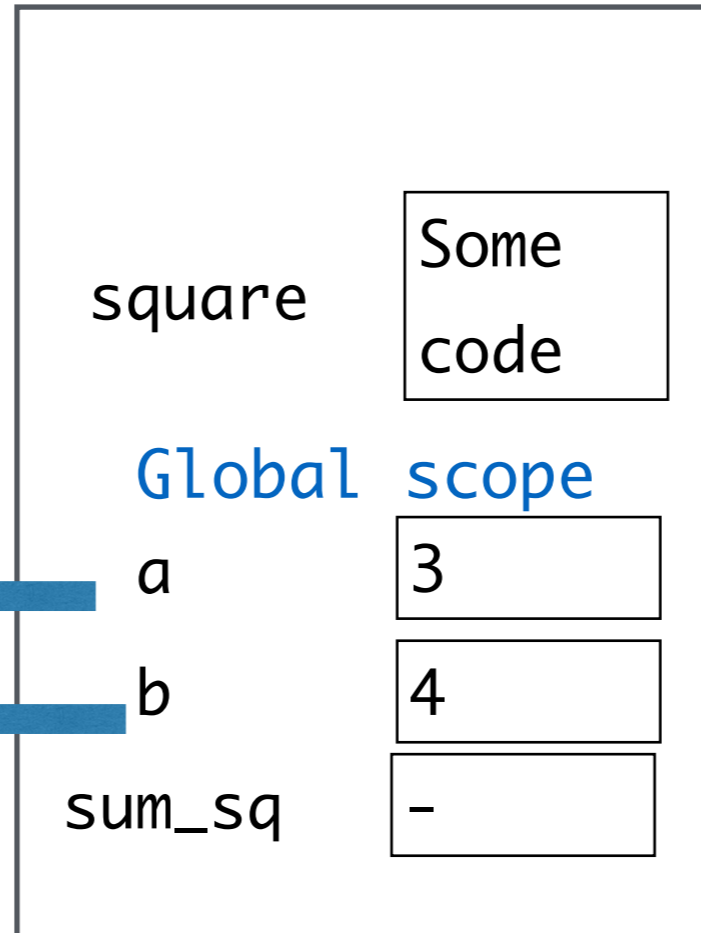


Understanding Scope

```
a = 3  
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = square(3) +  
         square(b) ?
```

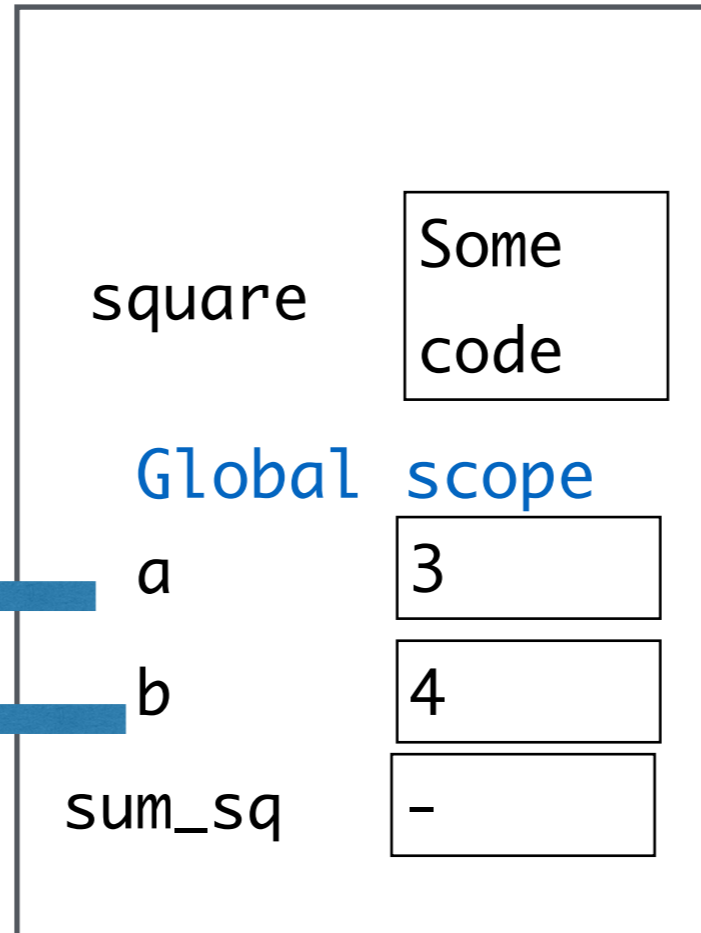


Understanding Scope

```
a = 3  
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = square(3) +  
         square(4) ?
```

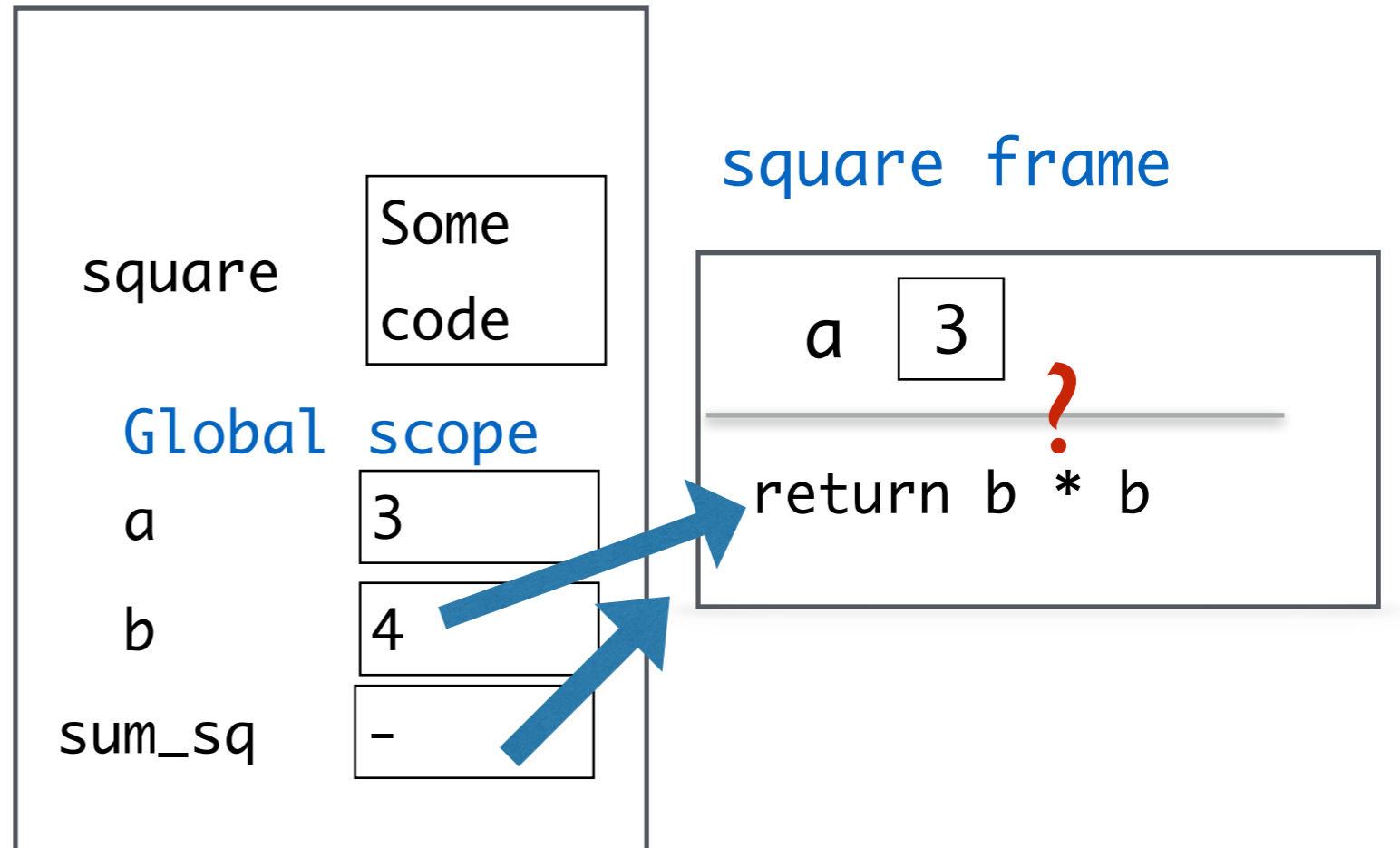


Understanding Scope

```
a = 3  
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = square(3) +  
         square(4)
```

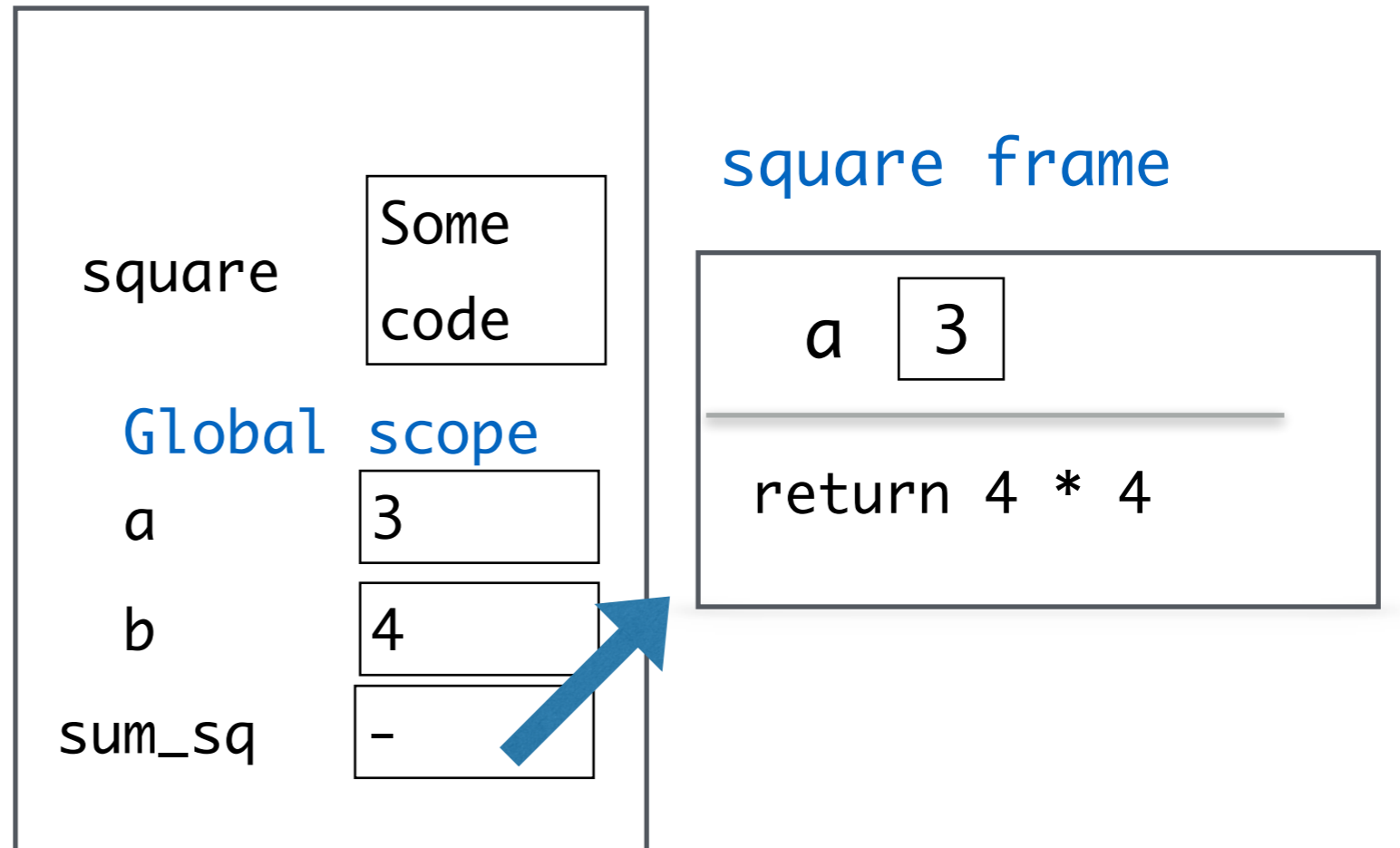


Understanding Scope

```
a = 3  
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = square(3) +  
         square(4)
```



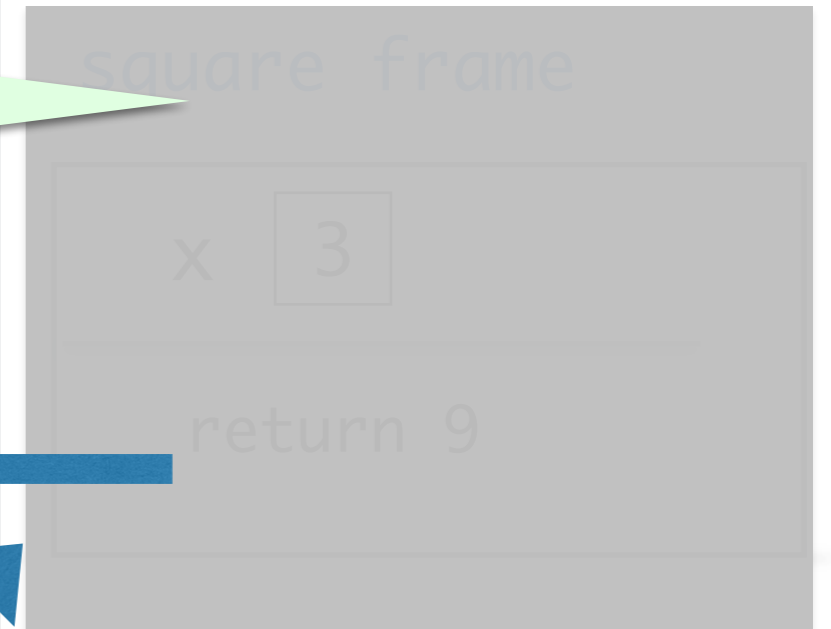
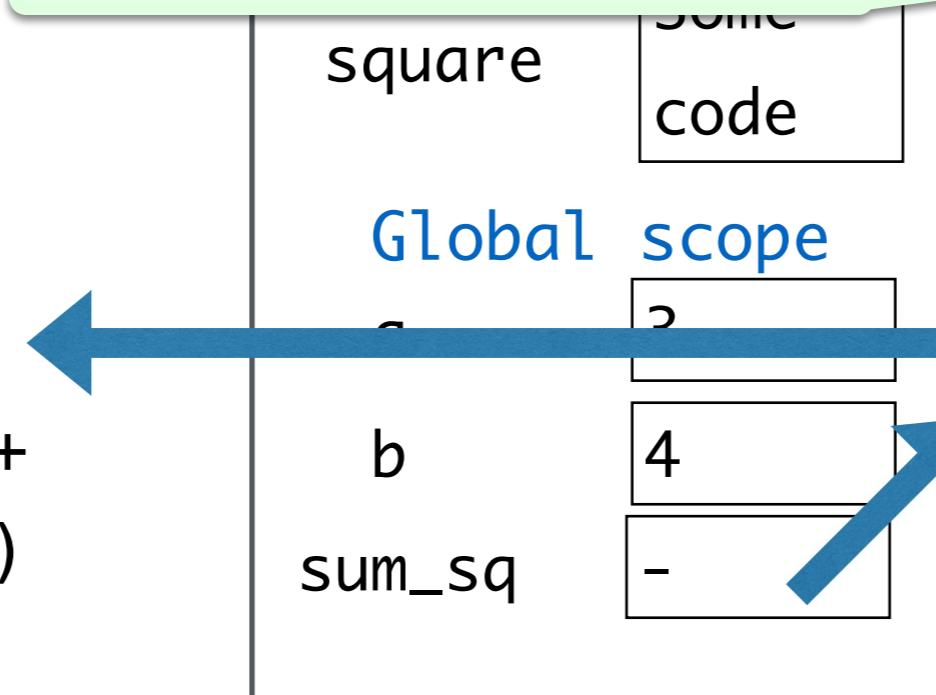
Understanding Scope

```
a = 3  
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = square(3) +  
         square(4)
```

Function frame destroyed (and all local variables lost) after return from function call

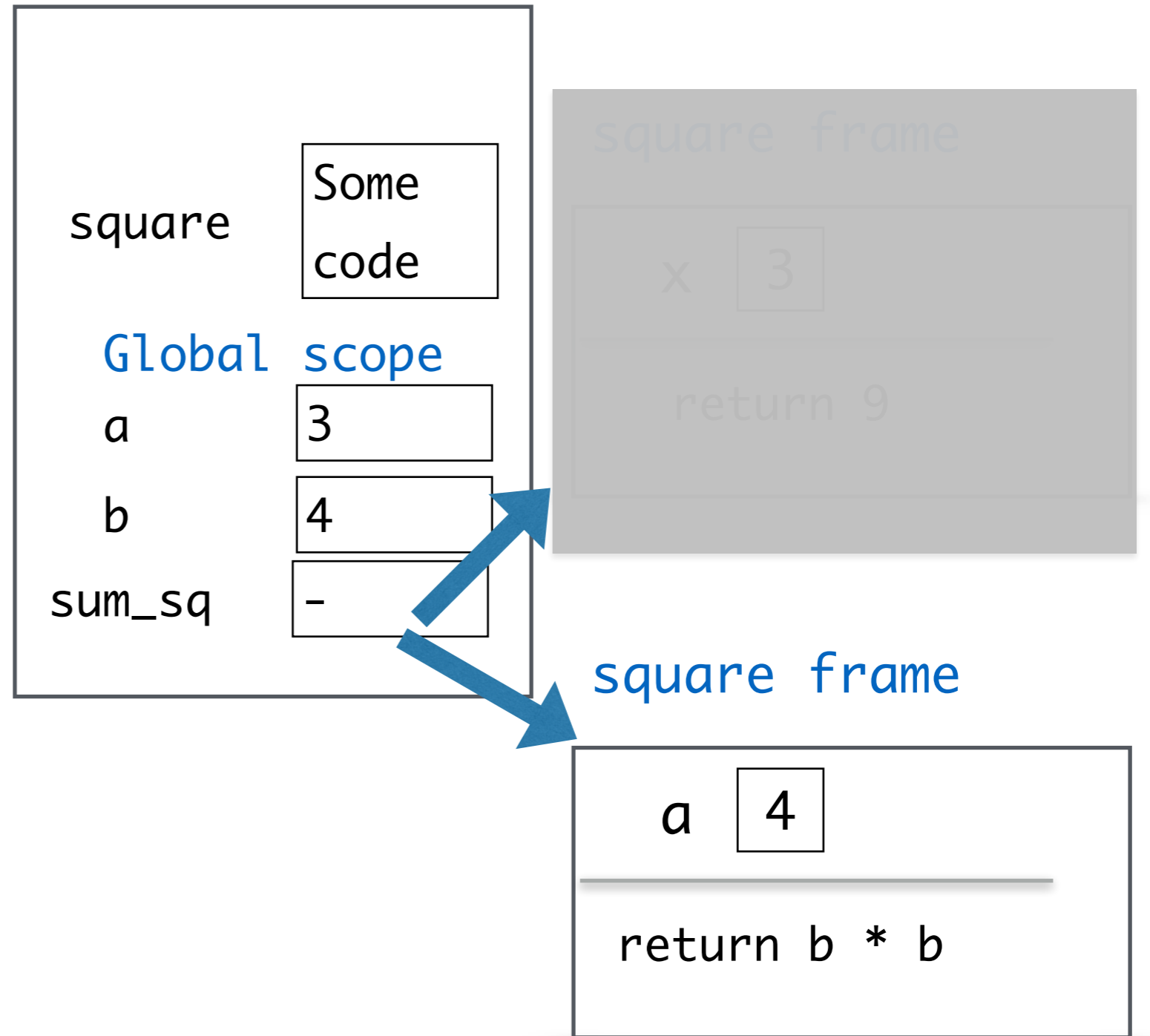


Understanding Scope

```
a = 3  
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = 16 +  
    square(4)
```

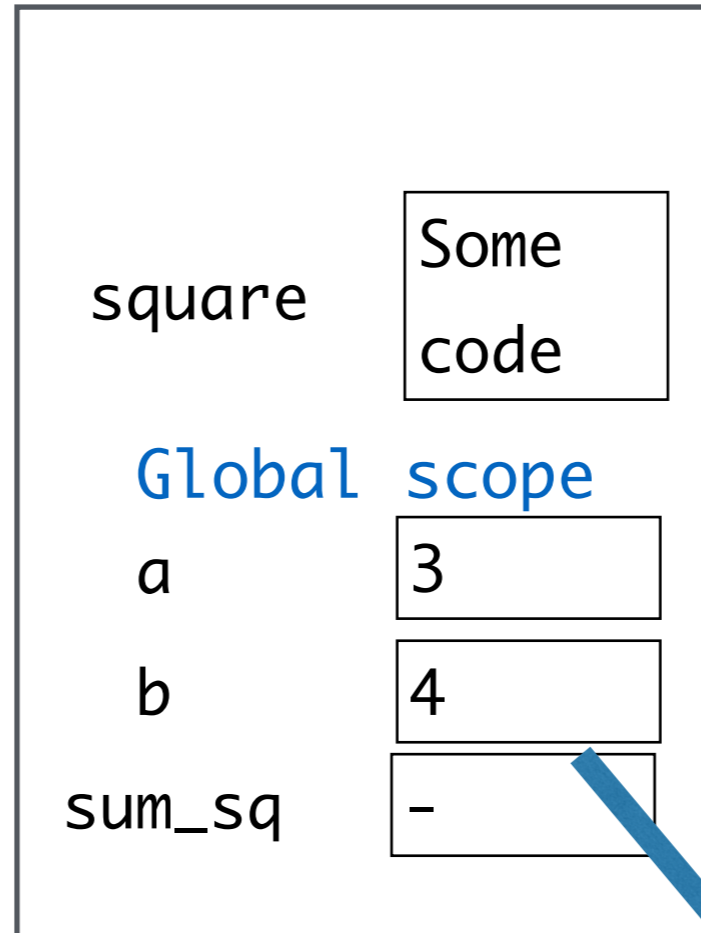


Understanding Scope

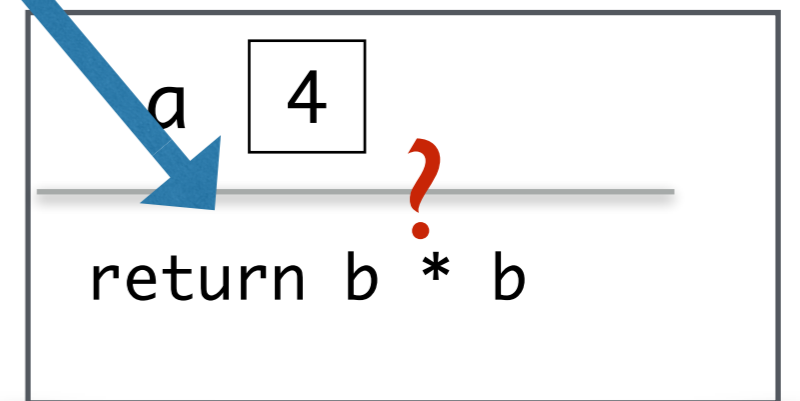
```
a = 3  
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = 16 +  
    square(4)
```



square frame

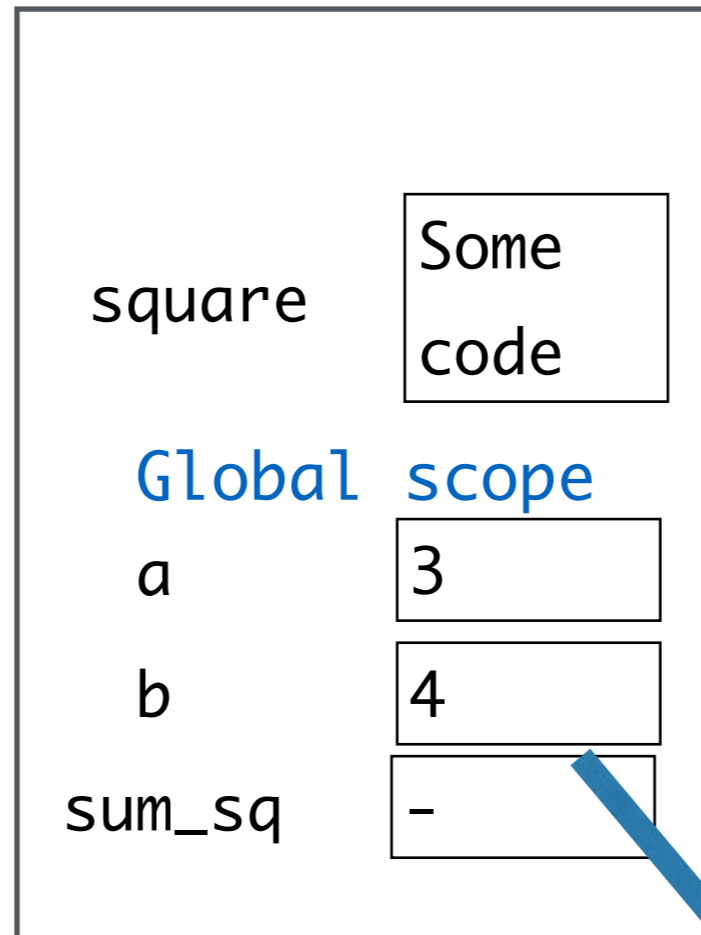


Understanding Scope

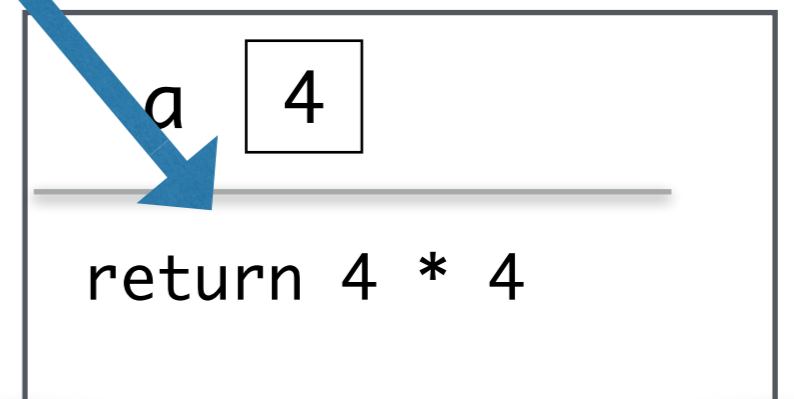
```
a = 3  
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = 16 +  
    square(4)
```



square frame



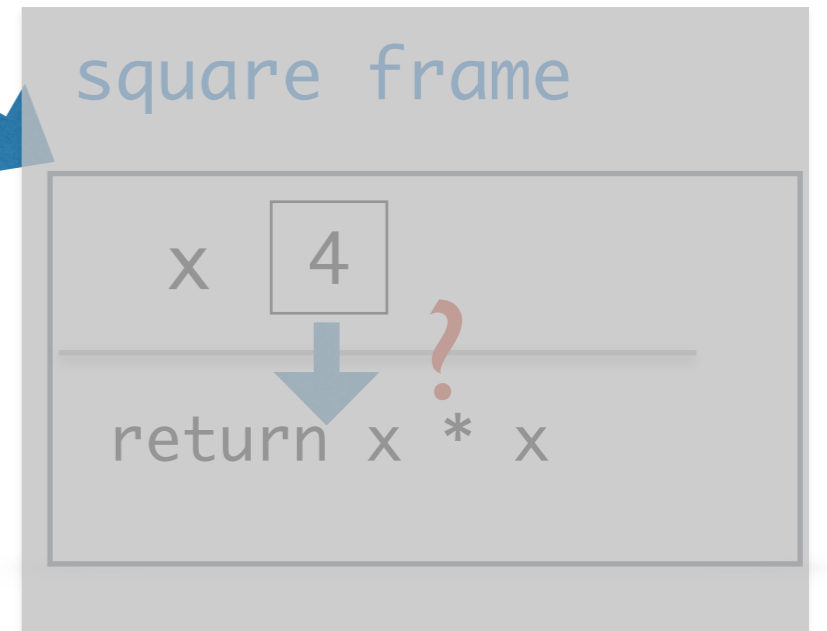
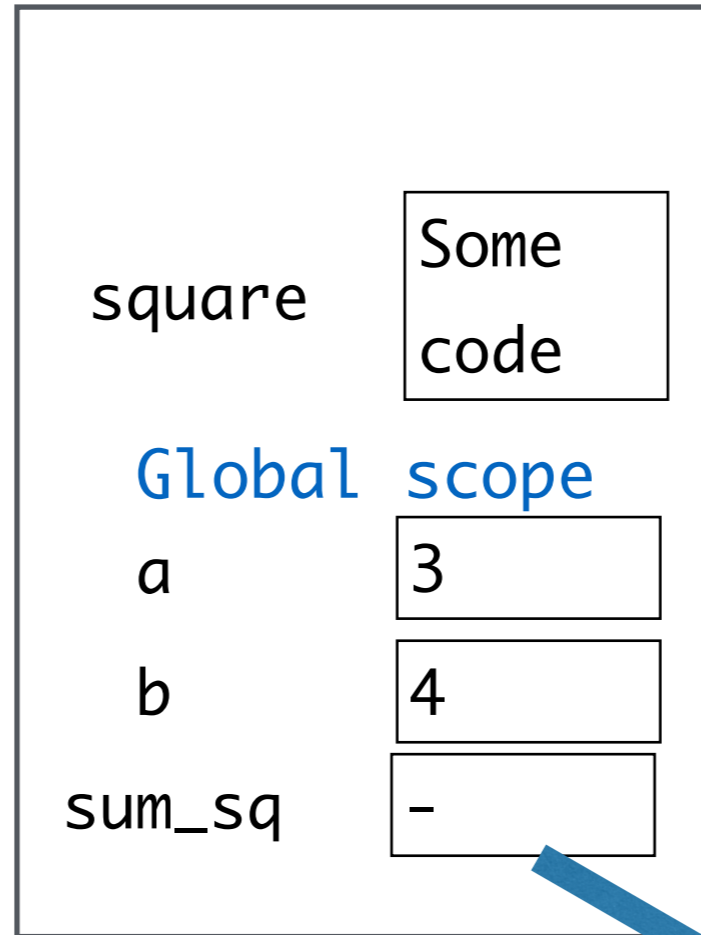
Understanding Scope

```
a = 3  
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = 16 +
```

```
square(4)
```



Understanding Scope

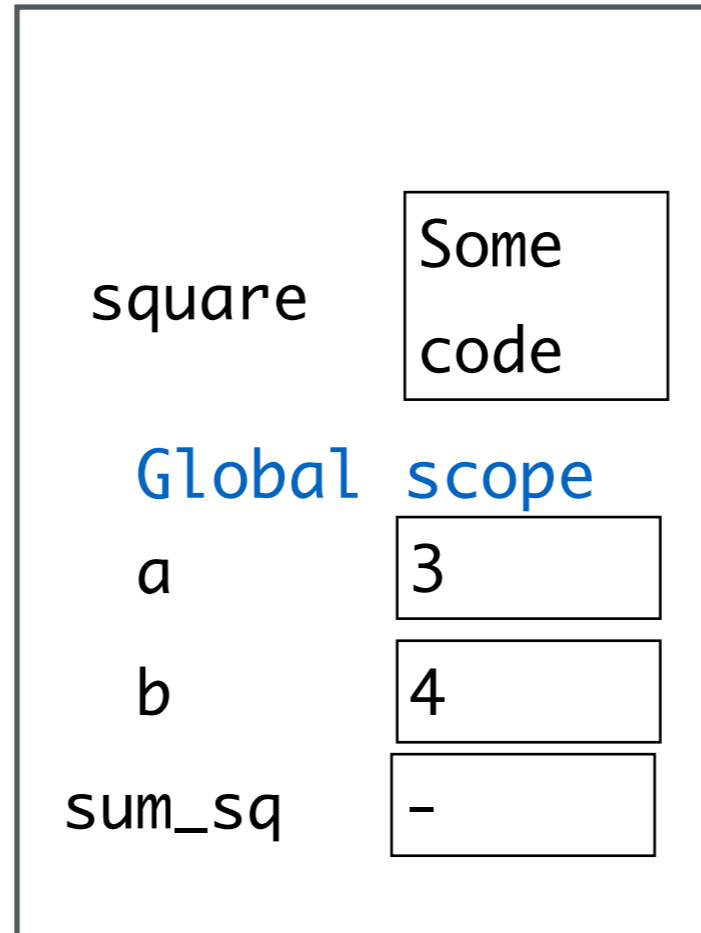
```
a = 3
```

```
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = 16 +
```

```
16
```



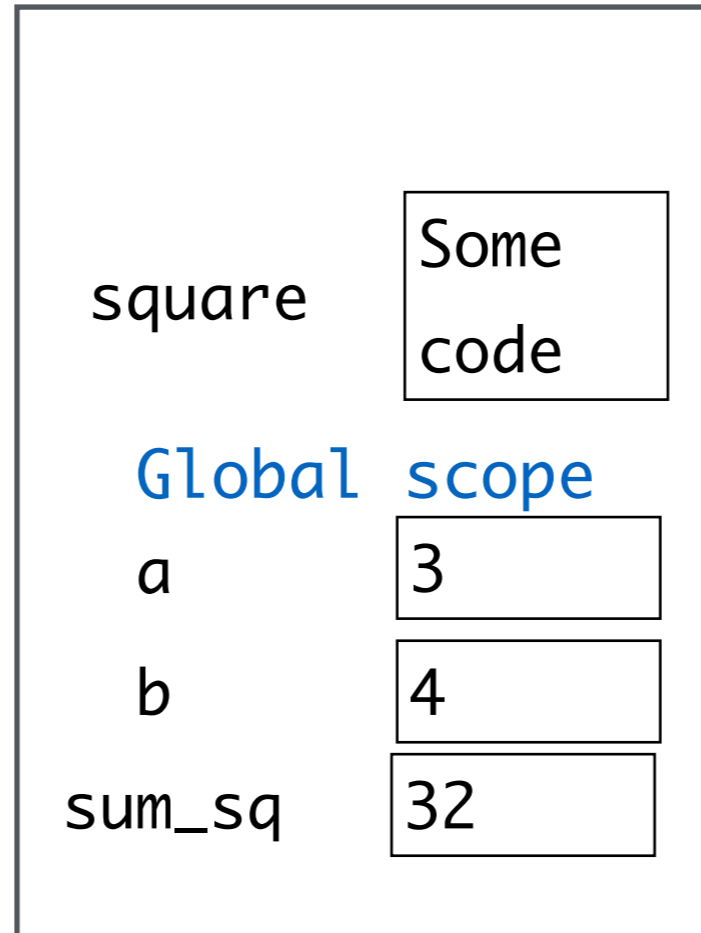
Understanding Scope

```
a = 3
```

```
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = 32
```



Understanding Scope

```
a = 3
```

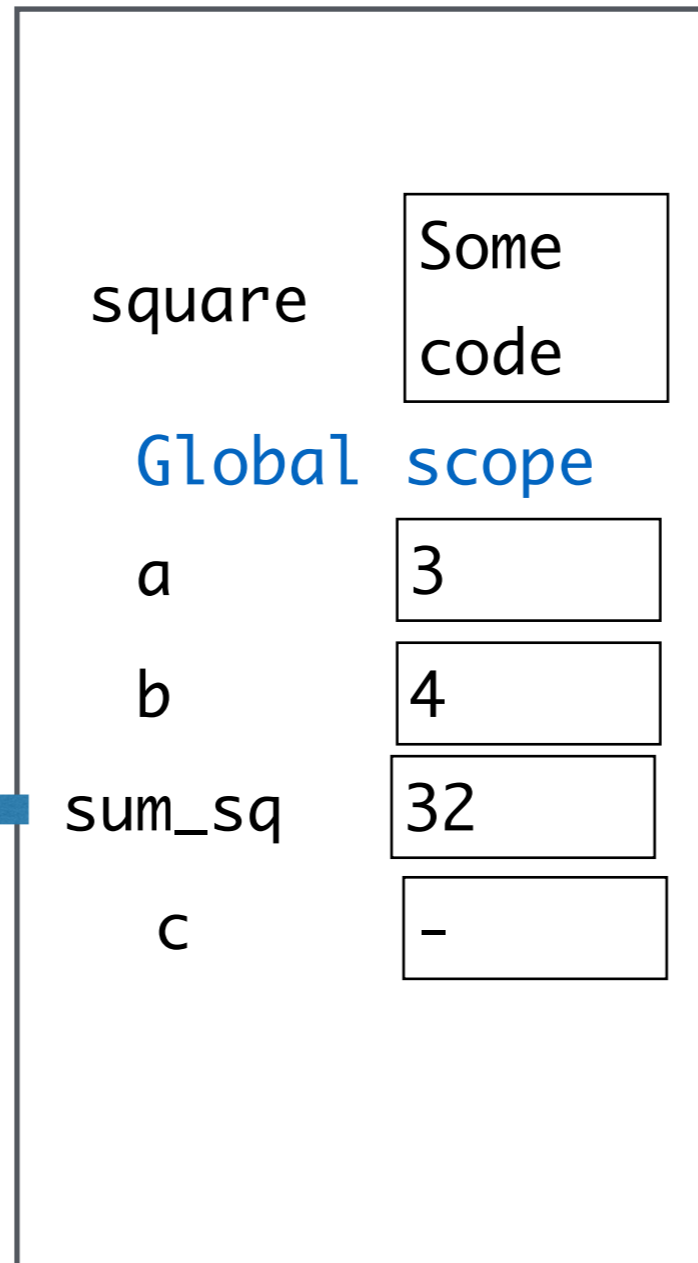
```
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = 32
```

```
c = sum_sq ** 0.5
```

```
print(c)
```



Understanding Scope

```
a = 3
```

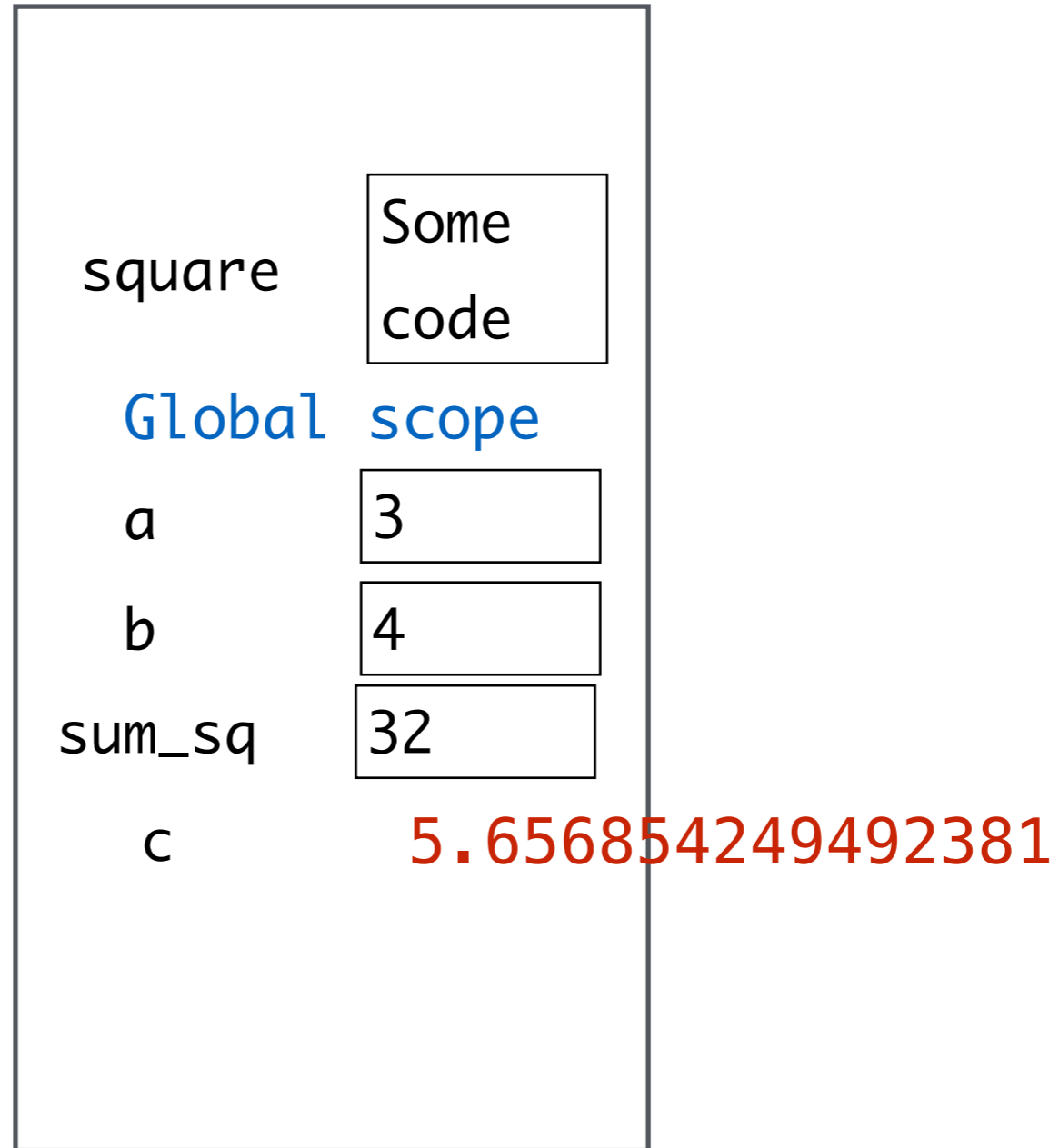
```
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = 32
```

```
c = 32 ** 0.5
```

```
print(c)
```



Understanding Scope

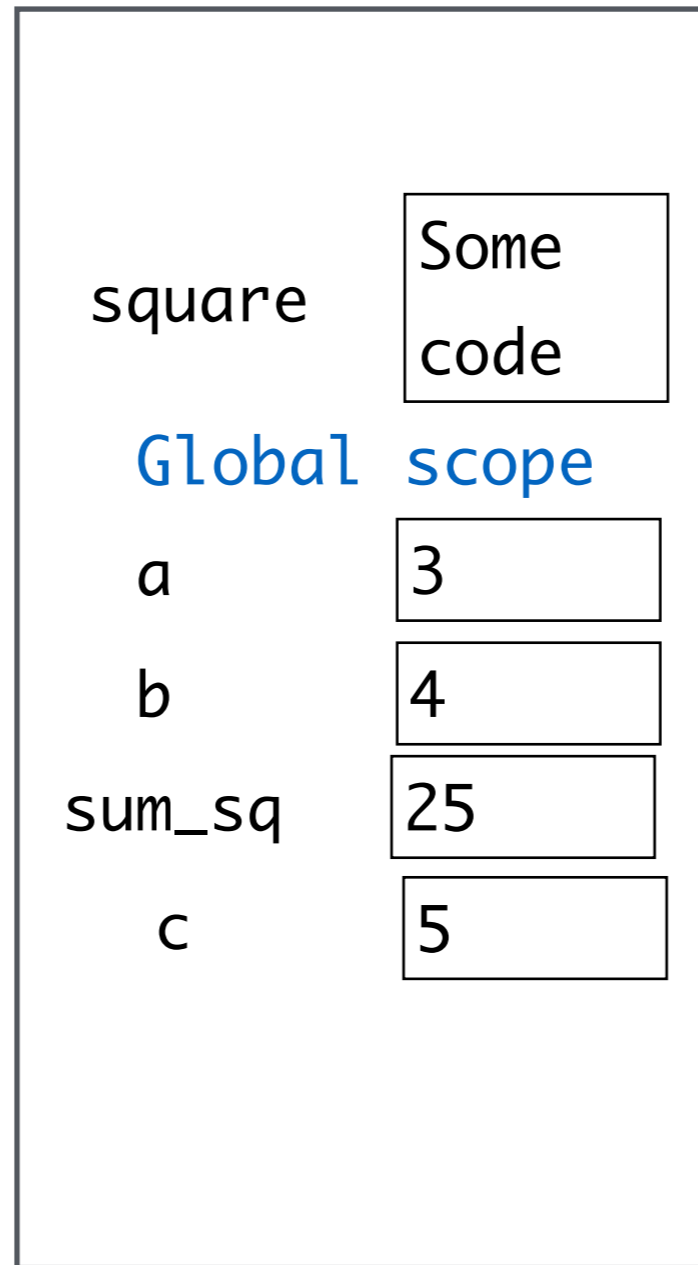
```
a = 3  
b = 4
```

```
def square(a):  
    return b * b
```

```
sum_sq = 32
```

```
c = 5.656854249492381
```

```
print(c)
```



Finally, **5.656854249492381** is printed

Takeaway: Local Before Global

When python encounters a new term, like a variable or function name, it *first looks locally*, before looking higher up.

If it can't find the value assigned to the term, you get a **NameError**.

More Examples in
Notebook

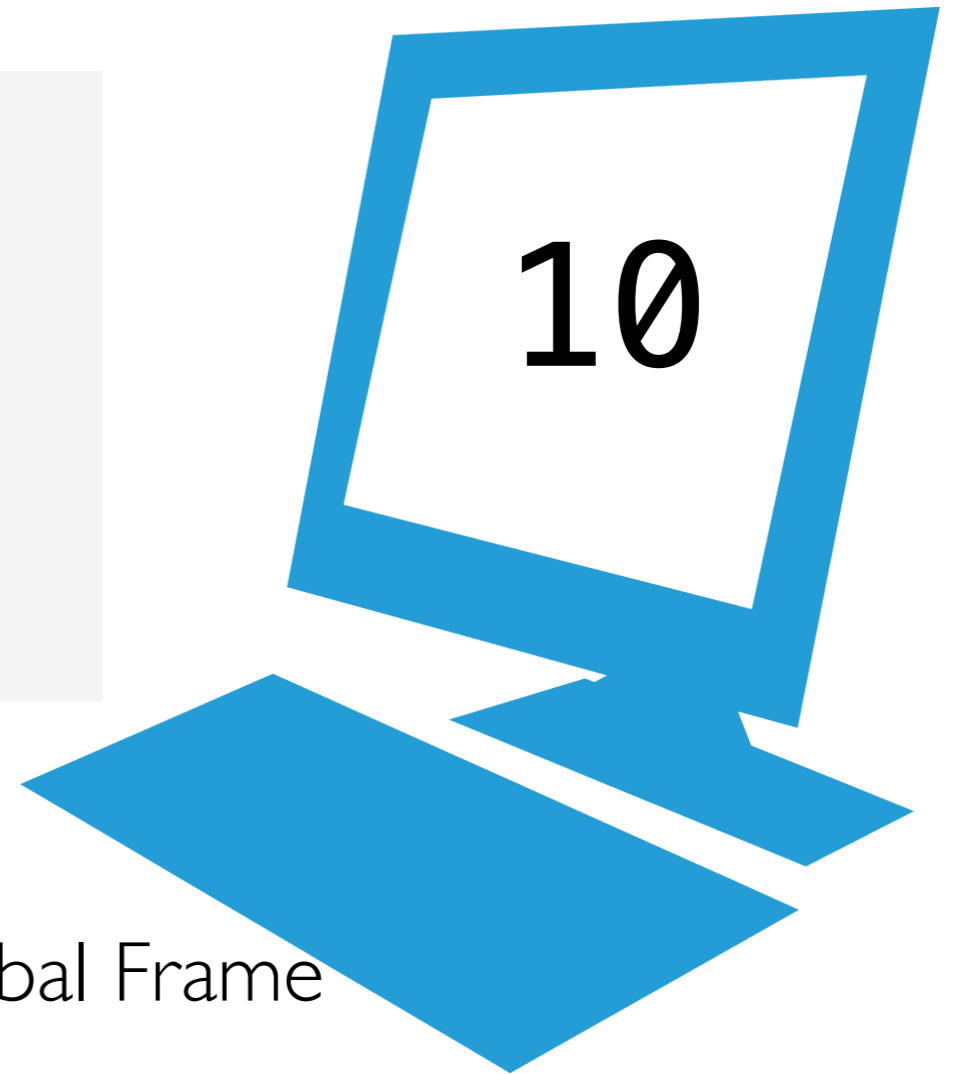
What gets printed to the screen?

```
multiplier = 3
def mystery(num):
    return multiplier * num
multiplier = 2
answer = mystery(5)
print(answer)
```



What gets printed to the screen?

```
multiplier = 3
def mystery(num):
    return multiplier * num
multiplier = 2
answer = mystery(5)
print(answer)
```



- `multiplier` is recorded as 3 on the Global Frame
- Then the `mystery()` blueprint is recorded on the Global Frame
- Then `multiplier` is re-assigned the value 2 on the Global Frame
- ...


What gets printed to the screen?

```
list = 2468  
list_str = list("whoops")  
print(list, list_str)
```



What gets printed to the screen?

```
list = 2468
list_str = list("whoops")
print(list, list_str)
```



TypeError:
'list'
object is
not callable

- `list` is a python keyword, in the Global Frame
- `list = ...` reassigns the value of `list` in the Global Frame
 - It's no longer the keyword, it's now an integer object
- So you can't call `list(...)` as the built-in list-casting function!
- ...This is why we don't use python keywords as variable names.

Helpful Tool for Learning How python Executes Code

- <https://pythontutor.com/cp/composingprograms.html>

