# CSCI 121:
# Frames & Environments Part II
# Nesting Requires Parents

# Returning function objects

Suppose a function object is returned:

```
def make_adder(by_how_much):
    return (lambda x: x + by_how_much)

add1 = make_adder(1)
add5 = make_adder(5)
```

- ➡ The function object "remembers" its local frame.
- ➡ This is called its ***parent frame***.
- ➡ A function object is a "closure." This is the description of its code along with info about its parent frame.

***closure = code + context***

# Parent frames

When a ***def*** is executed at the top level, that function's function parent frame is the global frame.

When a ***def*** is executed locally, that function's parent frame is that active local frame.

That active frame is the ***context*** in which that def is executed.

A *reference* to that parent frame is stored with the function object.

That ***def*** code + parent frame reference = the closure.

# Parent frames

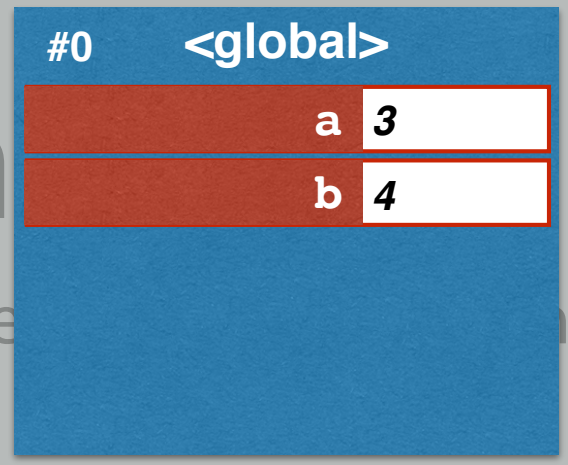
Let's revisit how Python seeks the global frame…

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the squares "
    print(s + str(a) + " " + str(b) + " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```

# Paren

Let's revisit how Python see ne…

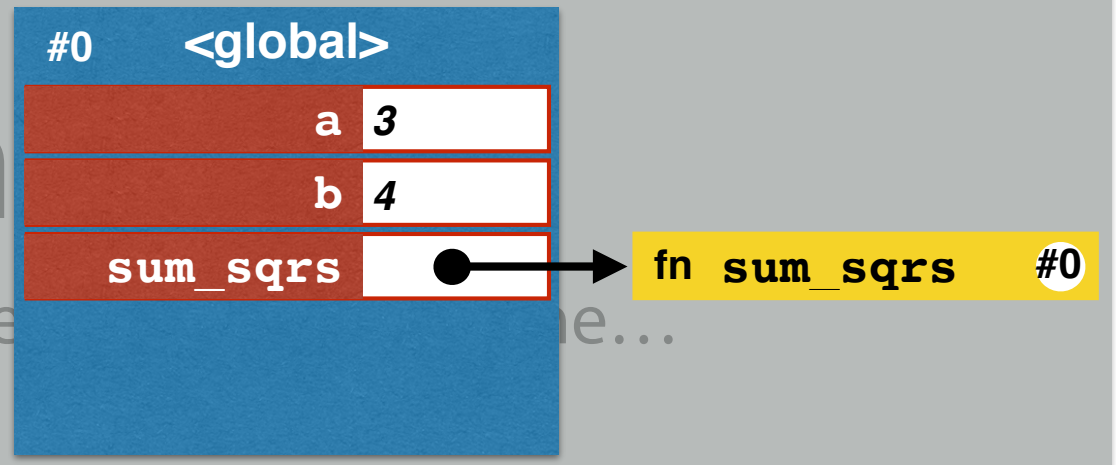
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the squares "
    print(s + str(a) + " " + str(b) + " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see ne…

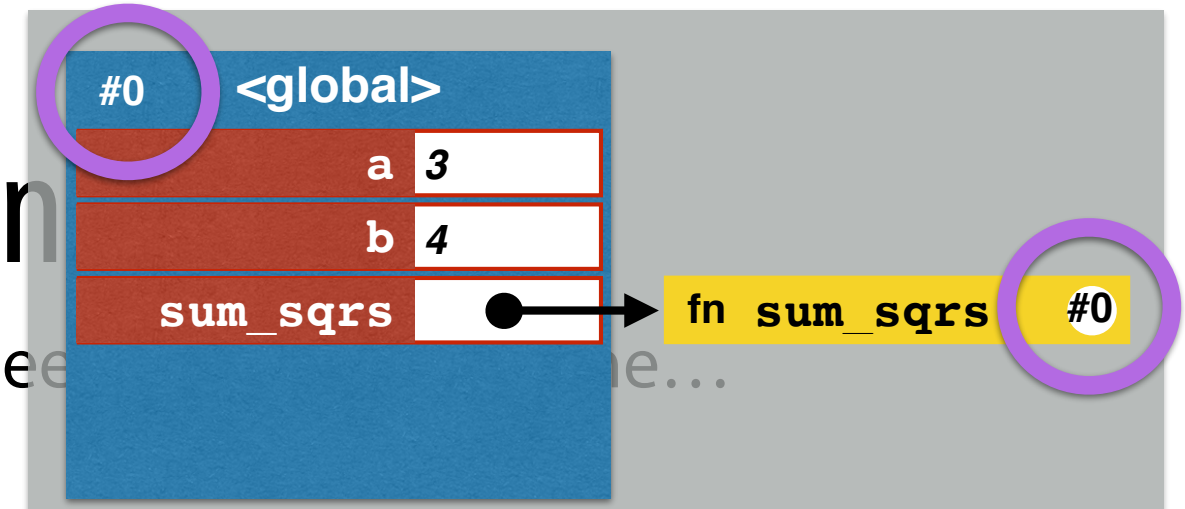
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the squares "
    print(s + str(a) + " " + str(b) + " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see ne…

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the squares "
    print(s + str(a) + " " + str(b) + " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```

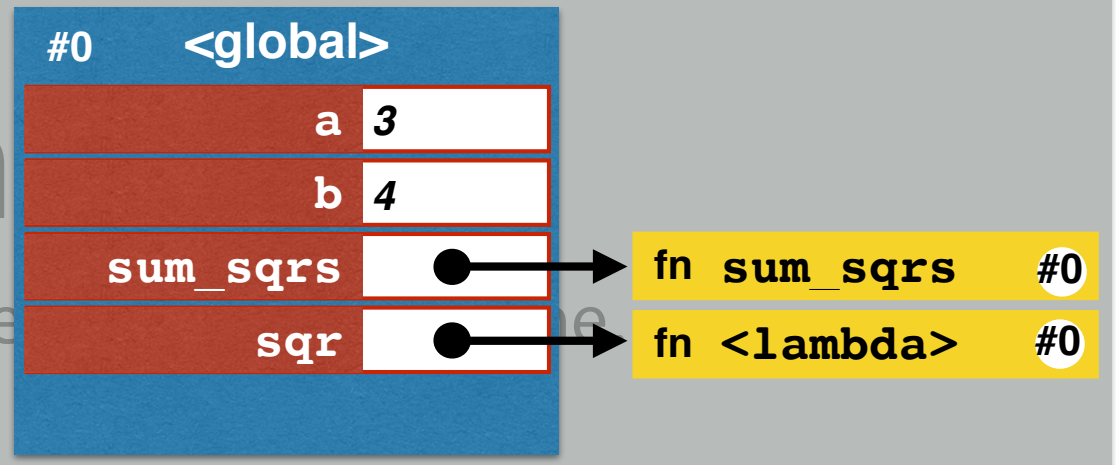



- The *def* of `sum_sqrs` occurs in the global context.
- This is `sum_sqrs`' parent frame.

# Paren

Let's revisit how Python see

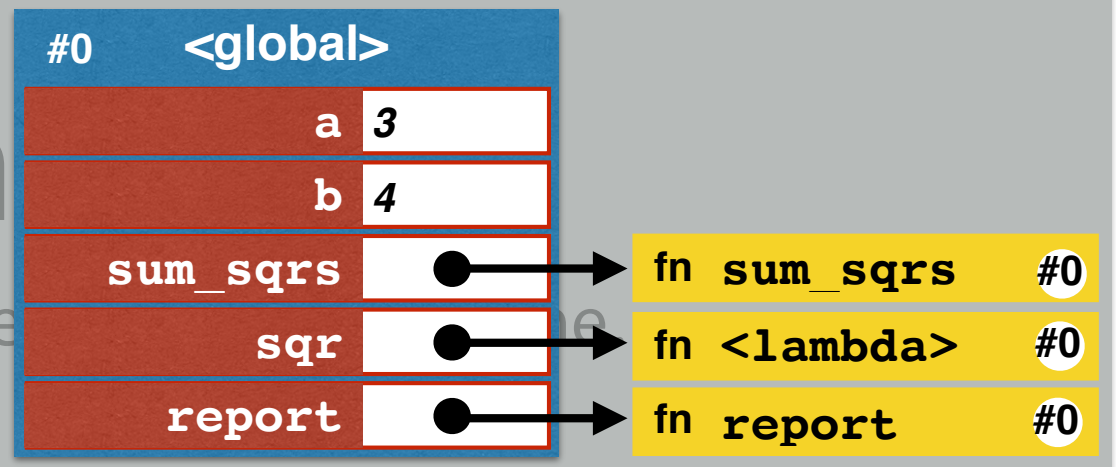
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the squares "
    print(s + str(a) + " " + str(b) + " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

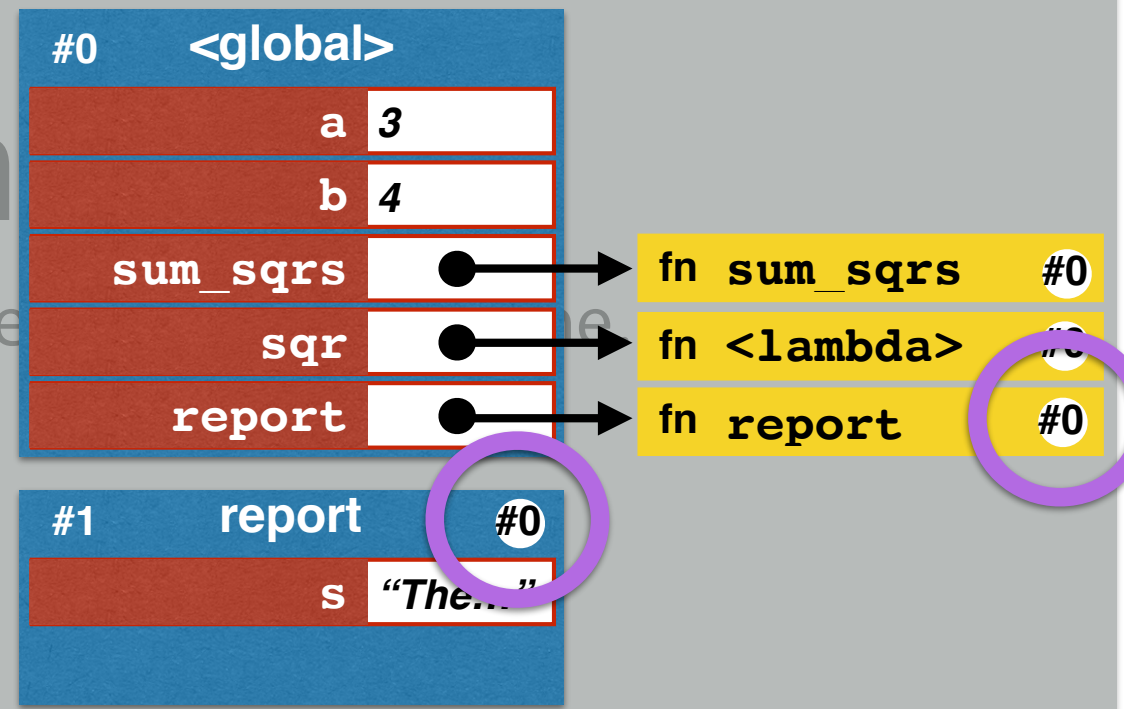
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the squares "
    print(s + str(a) + " " + str(b) + " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of th
    print(s + str(a) +
    r = sum_sqrs(a,b)
    print(r)
report()
```

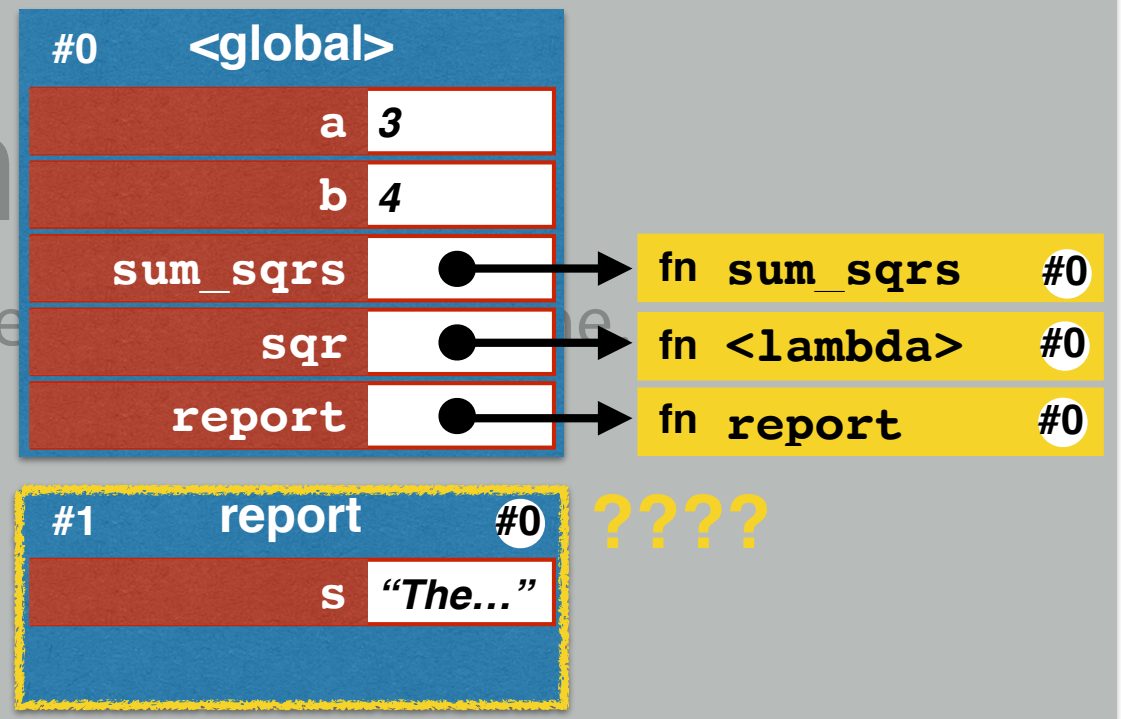



➡ The local frame for this call to `report` uses `report`'s parent frame.

# Paren

Let's revisit how Python see

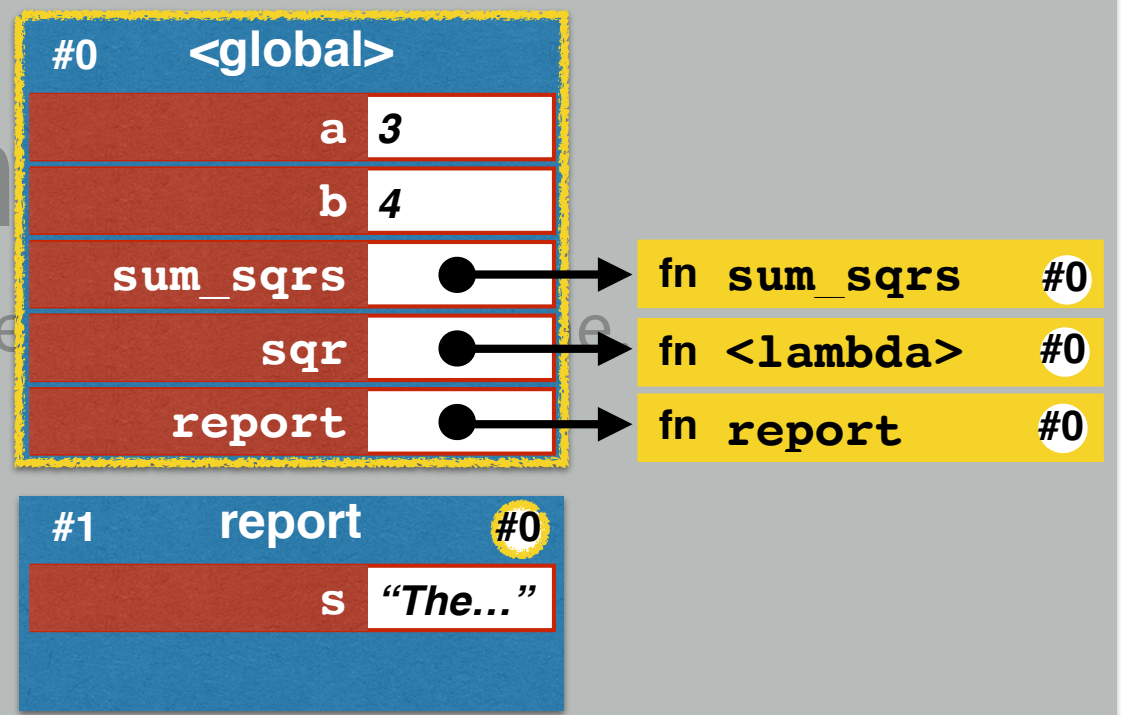
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the squares "
    print(s + str(a) + " " + str(b) + " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python se

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of th
    print(s + str(a) + " " + str(b) + " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```

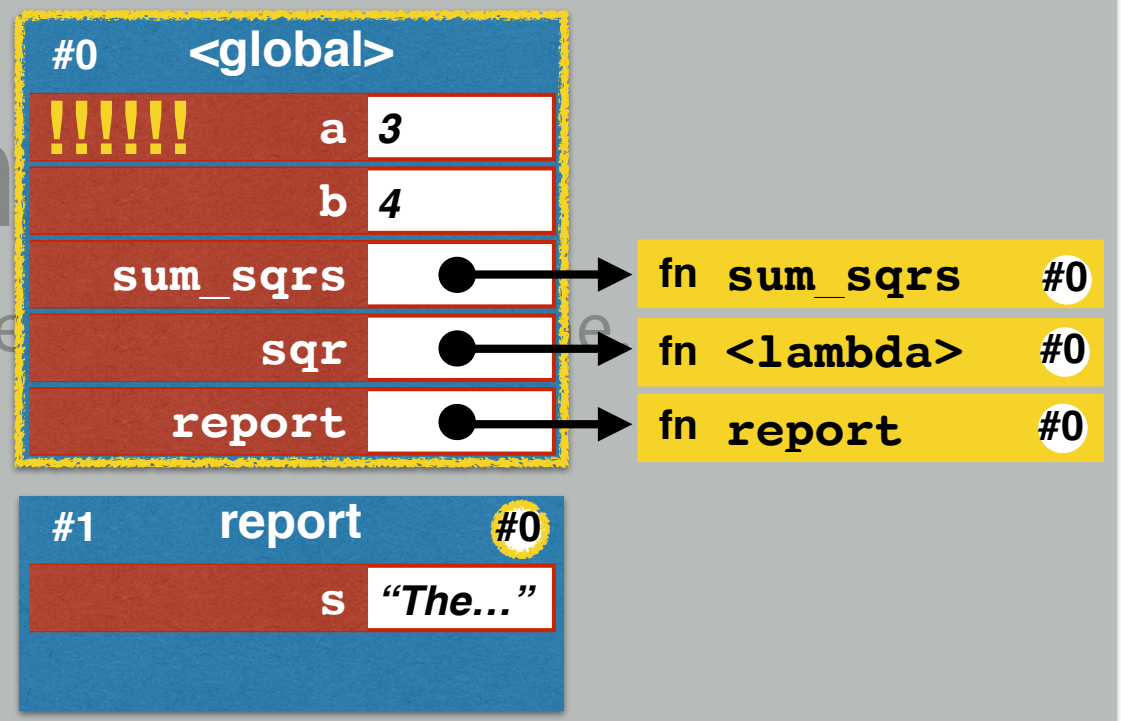



➡ Since a is unknown within the local frame, Python checks the parent frame.

# Paren

Let's revisit how Python see

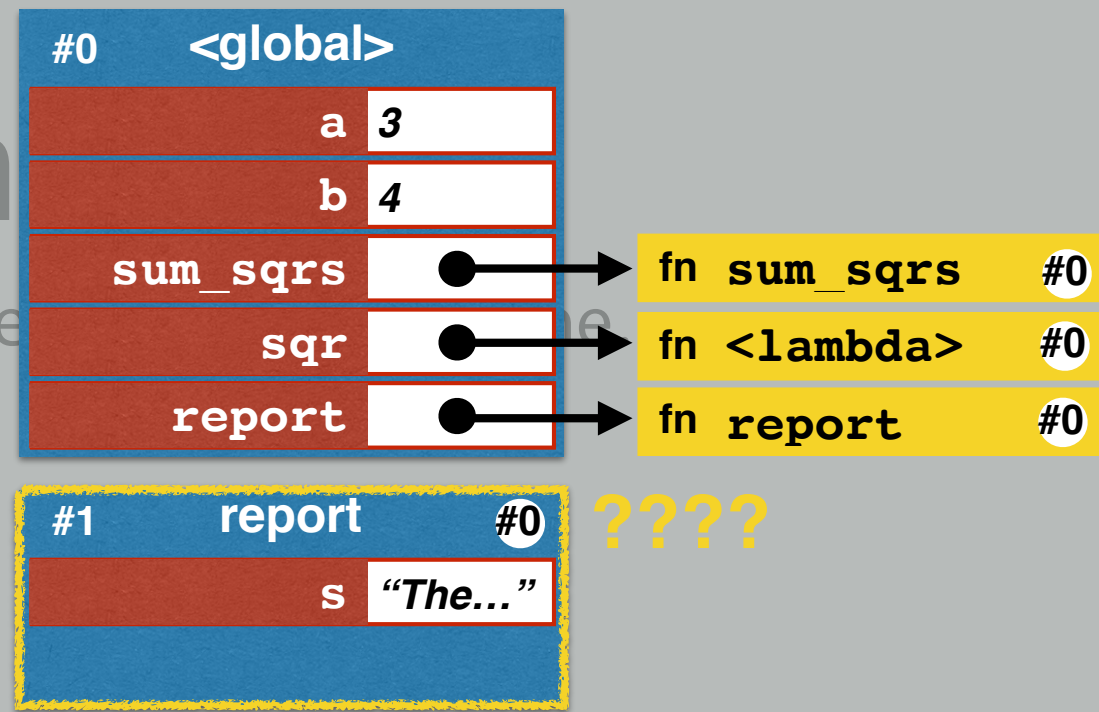
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the squares "
    print(s + str(a) + " " + str(b) + " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

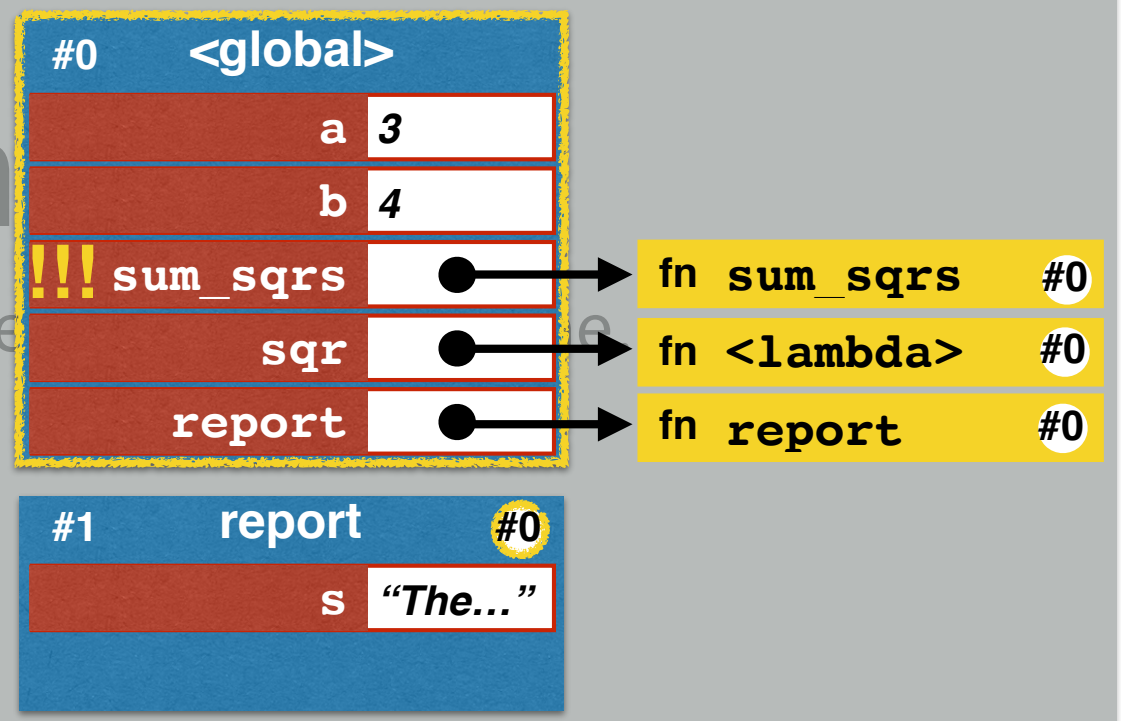
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the squares "
    print(s + str(a) + " " + str(b) + " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

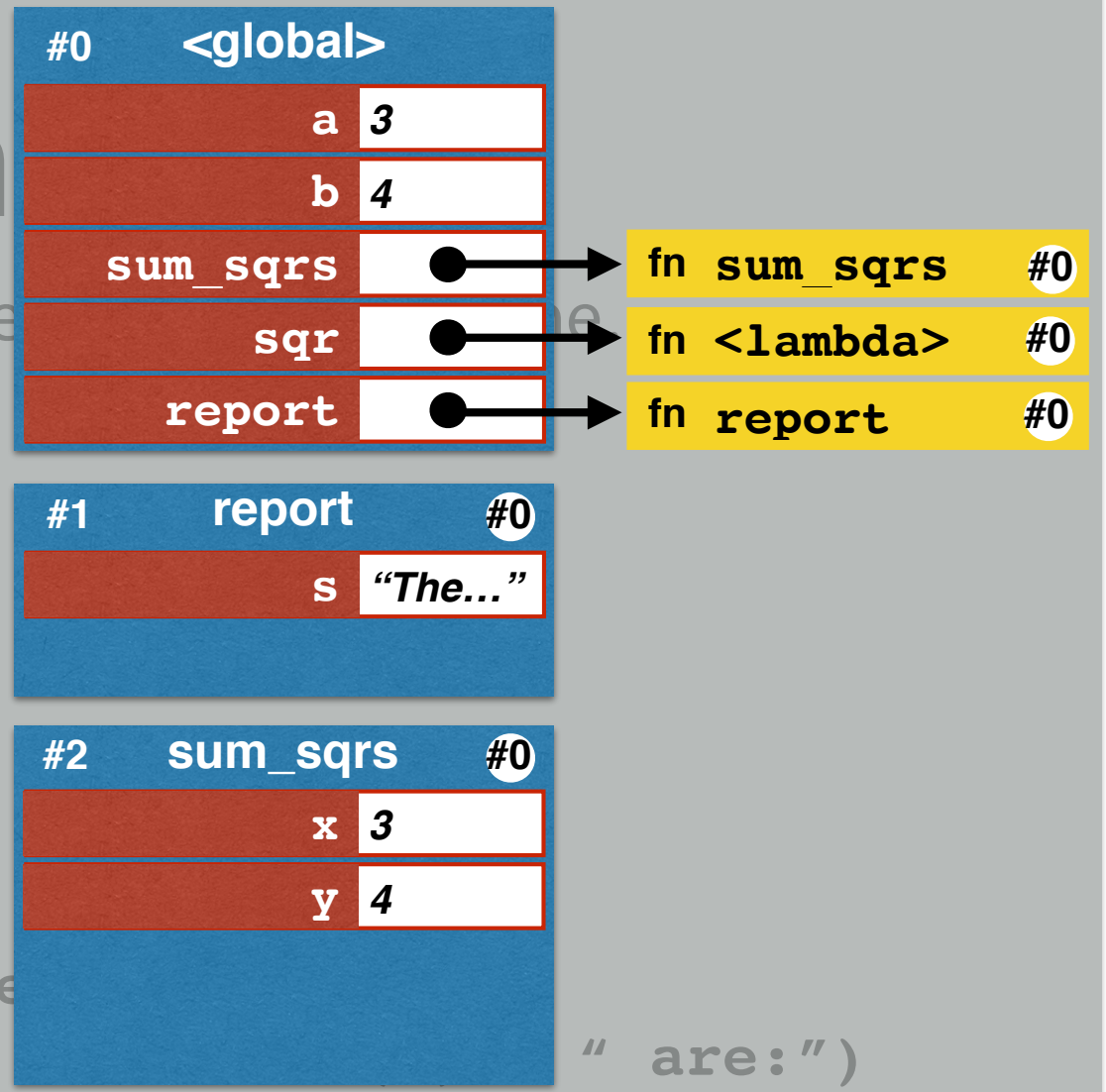
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the squares "
    print(s + str(a) + " " + str(b) + " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

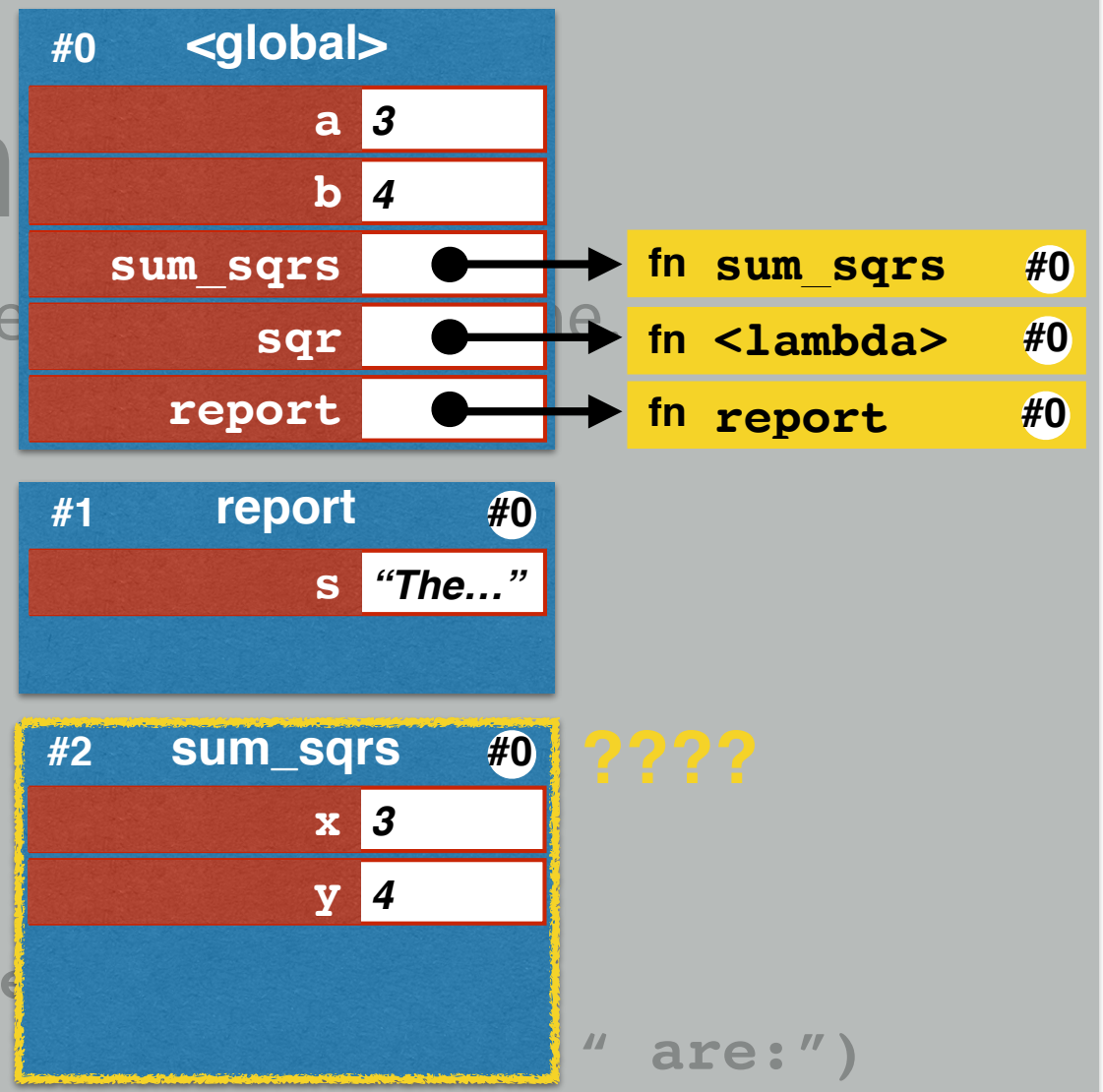
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the
    print(s + str(a) +                    " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

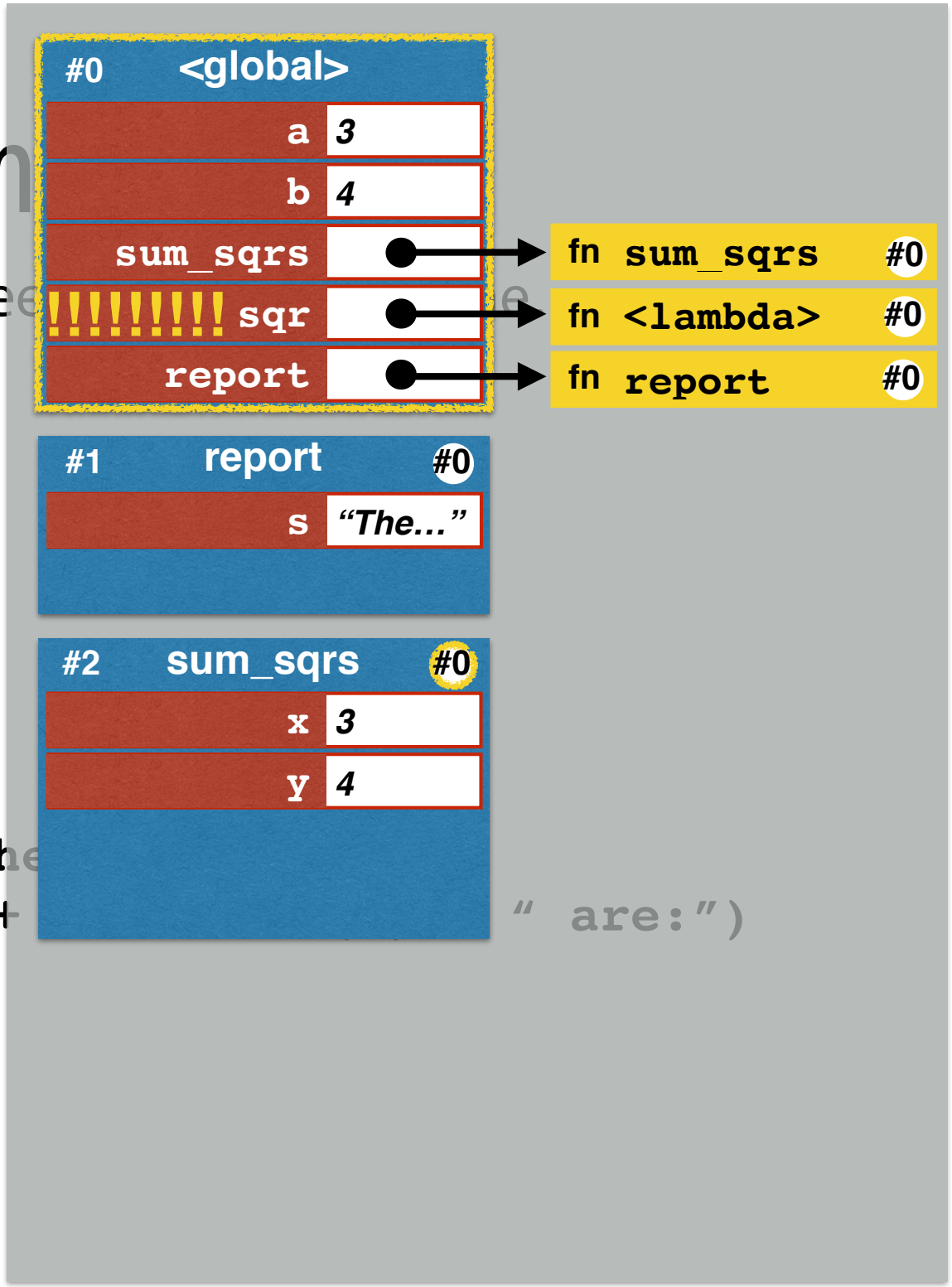
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the
    print(s + str(a) +                    " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

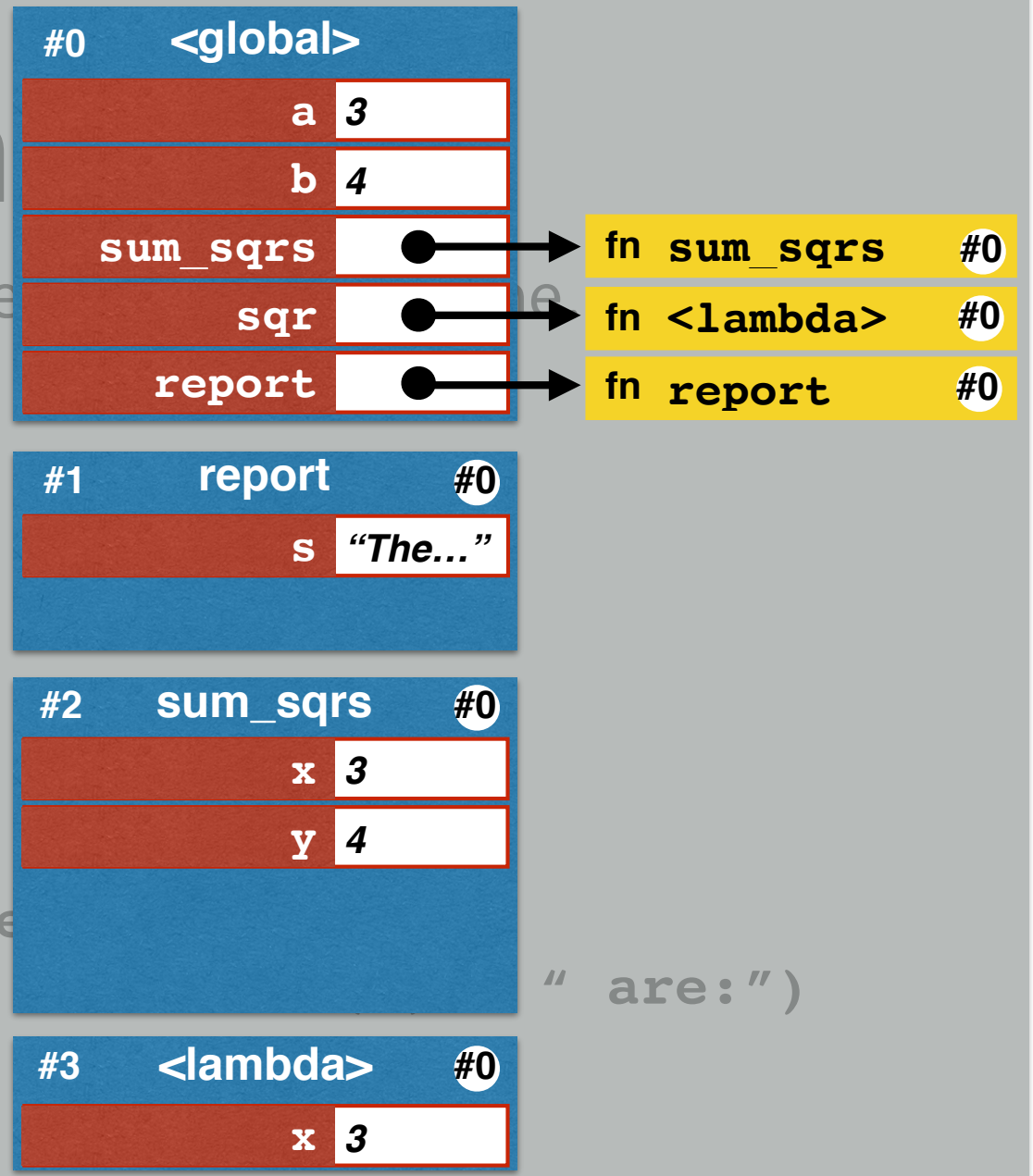
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the
    print(s + str(a) +                    " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

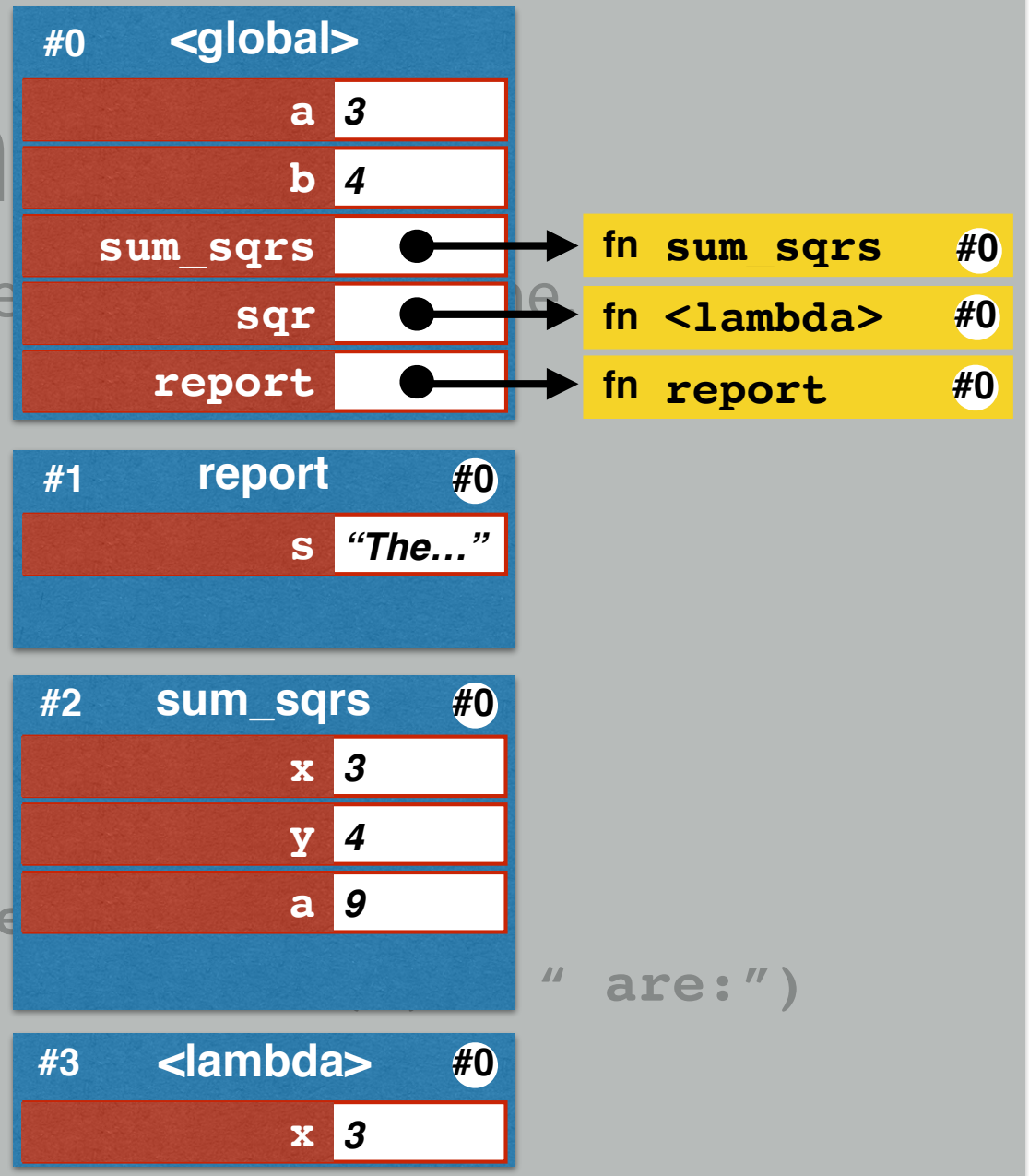
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the
    print(s + str(a) +                    " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

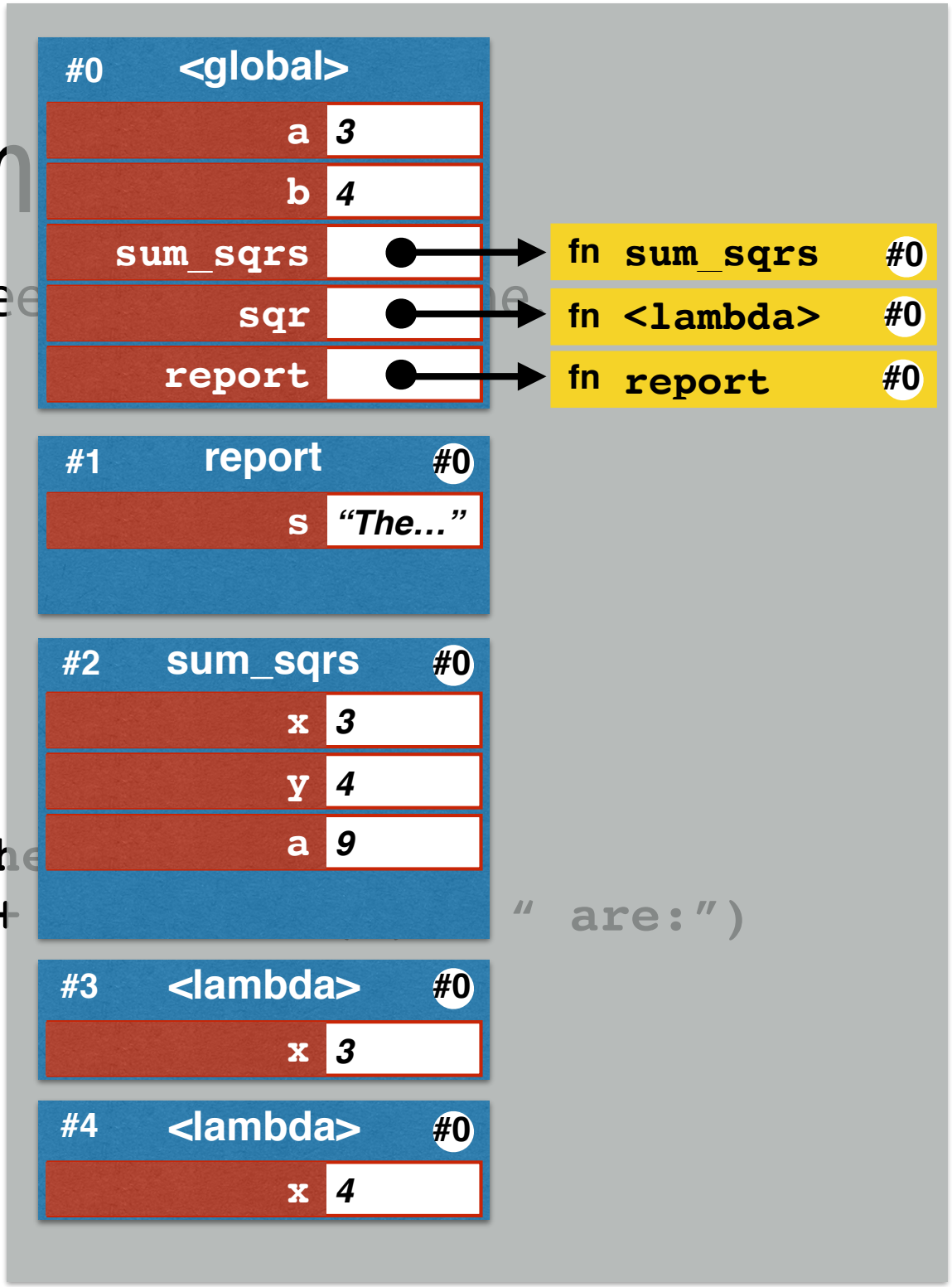
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the
    print(s + str(a) +                    " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

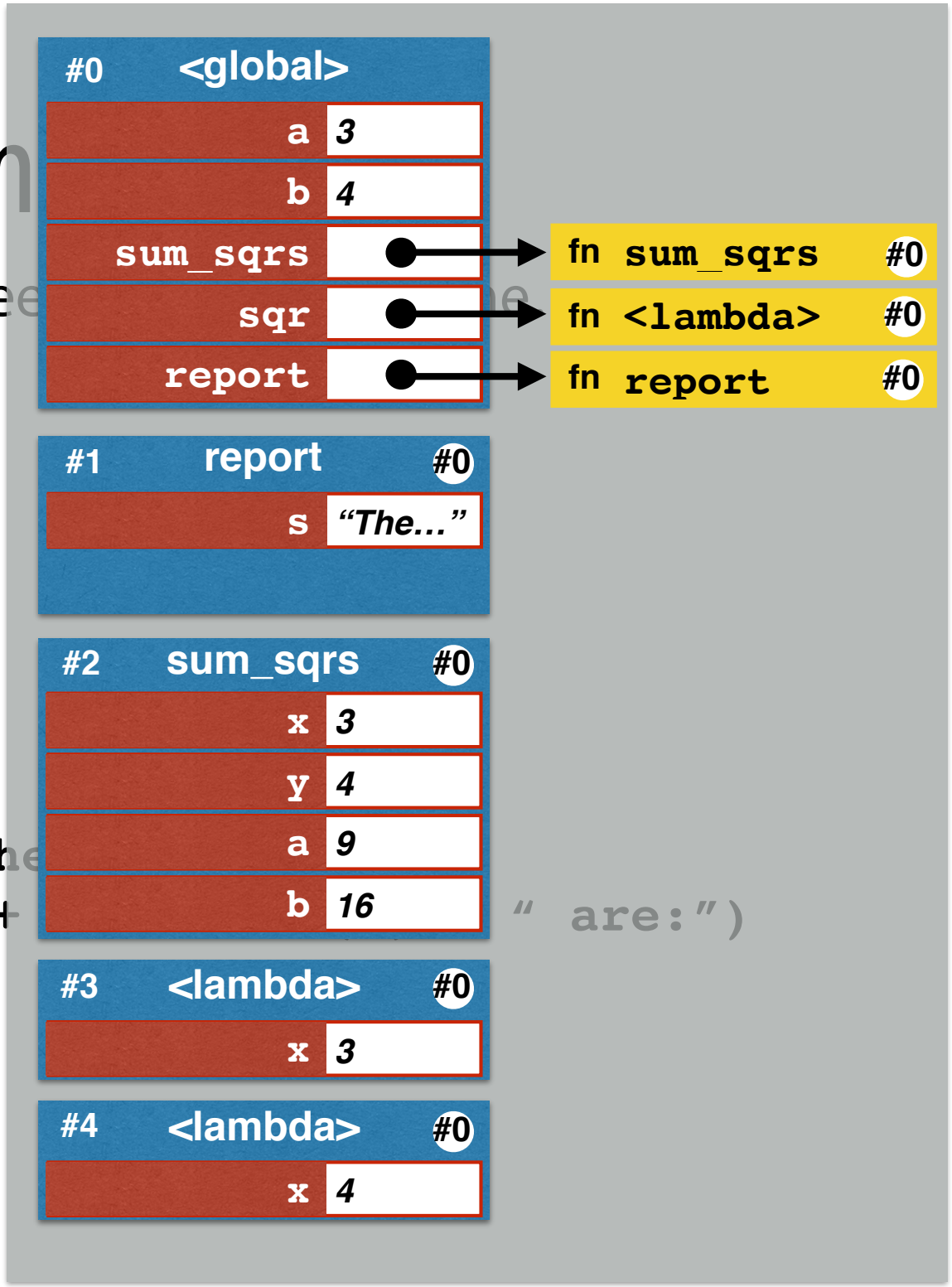
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the
    print(s + str(a) +                    " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

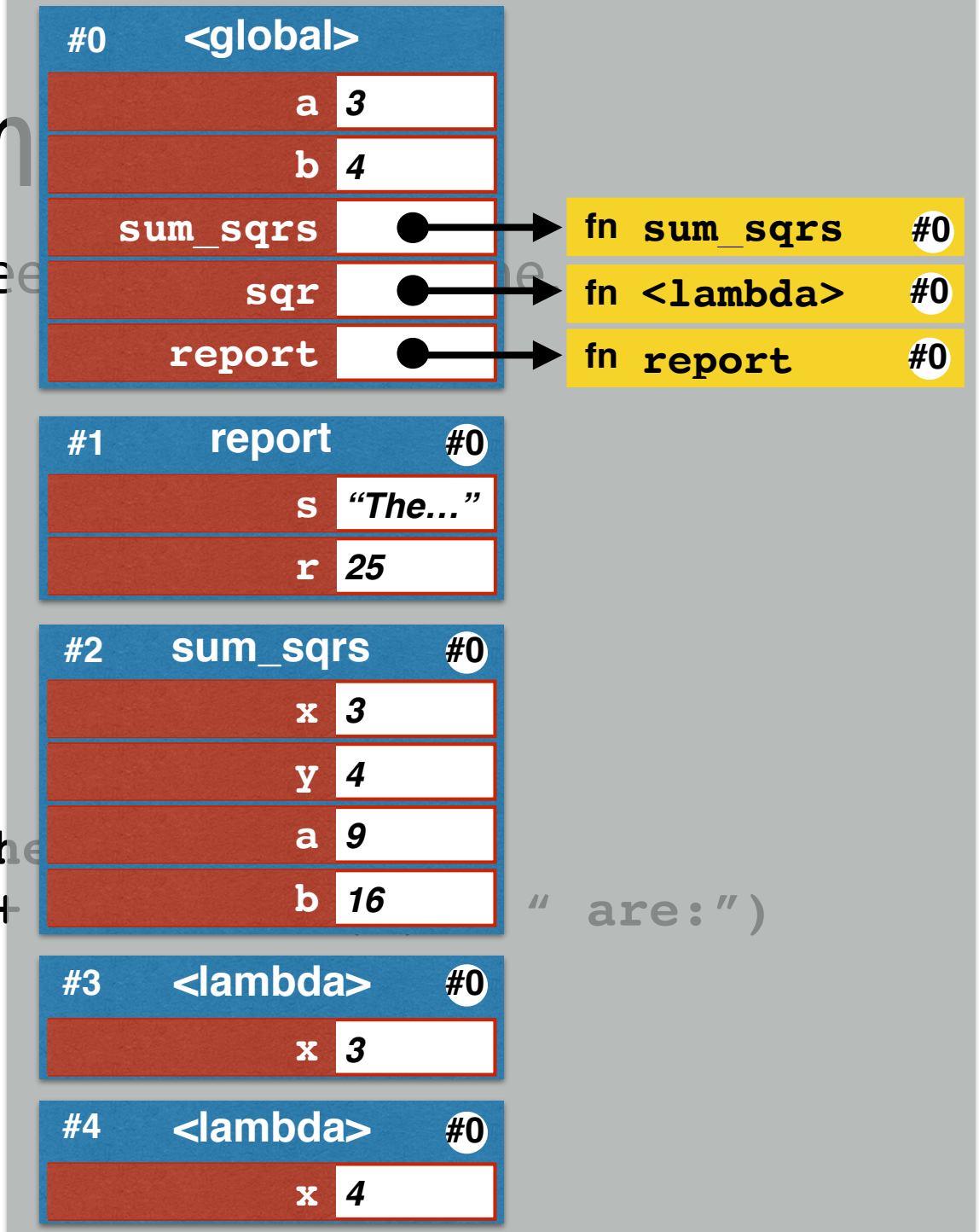
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the
    print(s + str(a) +                    " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

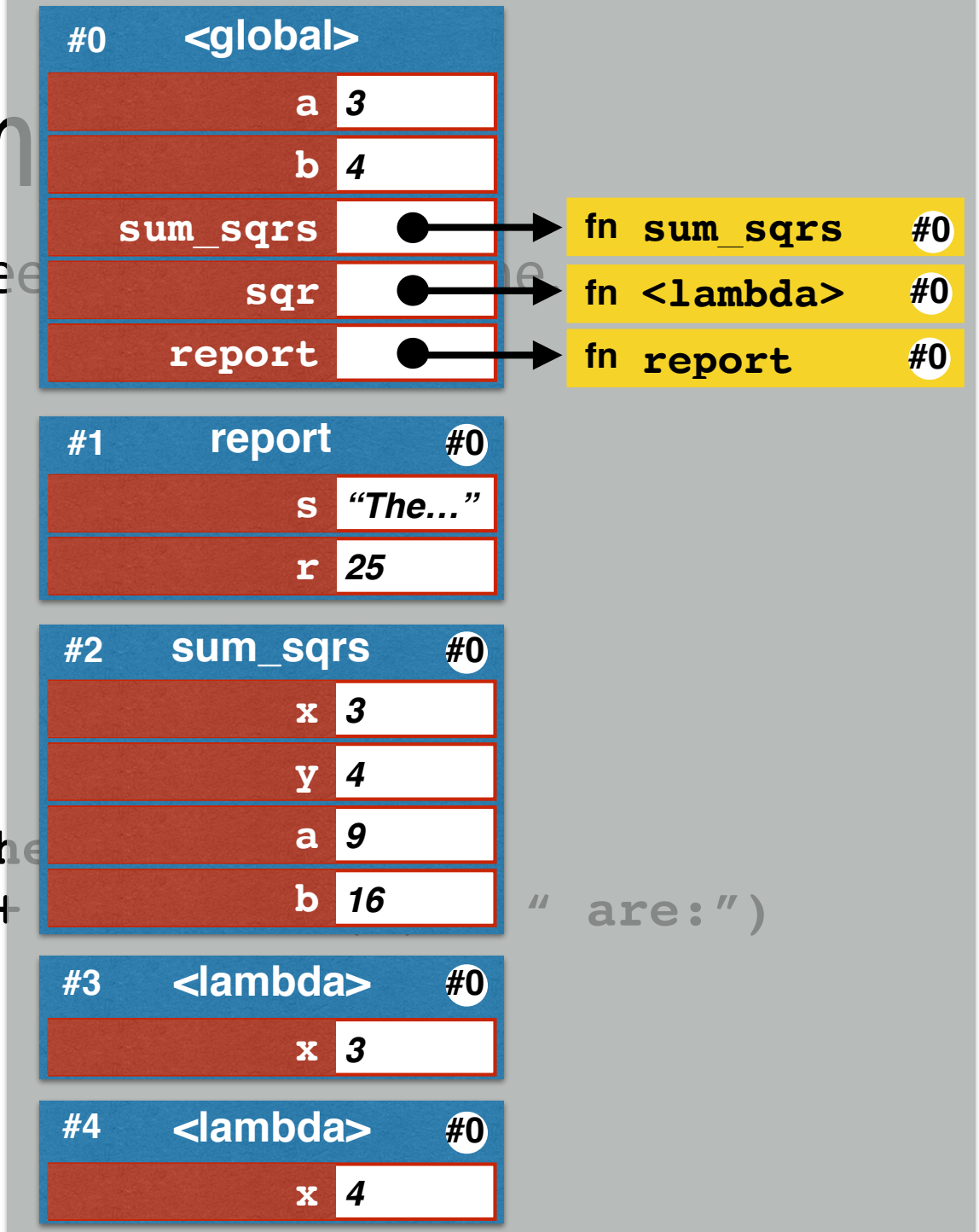
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the
    print(s + str(a) +                    " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Paren

Let's revisit how Python see

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = "The sum of the
    print(s + str(a) +                    " are:")
    r = sum_sqrs(a,b)
    print(r)
report()
```



# Returning function objects

Suppose a function object is returned:

```
def make_adder(by_how_much):
    return (lambda x: x + by_how_much)

add1 = make_adder(1)
add5 = make_adder(5)
```

- ➡ The function object "remembers" its local frame.
- ➡ This is called its ***parent frame***.
- ➡ A function object is a "closure." This is the description of its code along with info about its parent frame.

***closure = code + context***