

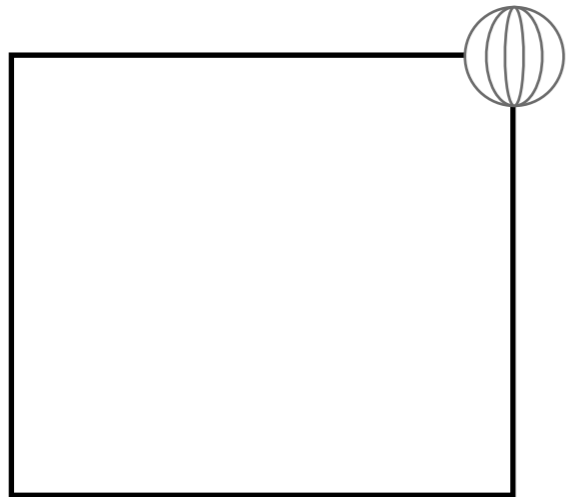
# 61A Lecture 4

---

Friday, September 2

# What Happened with def square(square)?

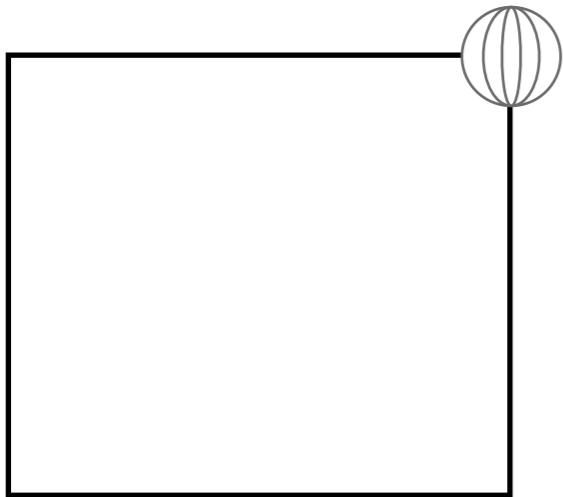
---



```
def square(square):  
    return mul(square, square)  
from operator import mul  
square(4)
```

# What Happened with def square(square)?

---

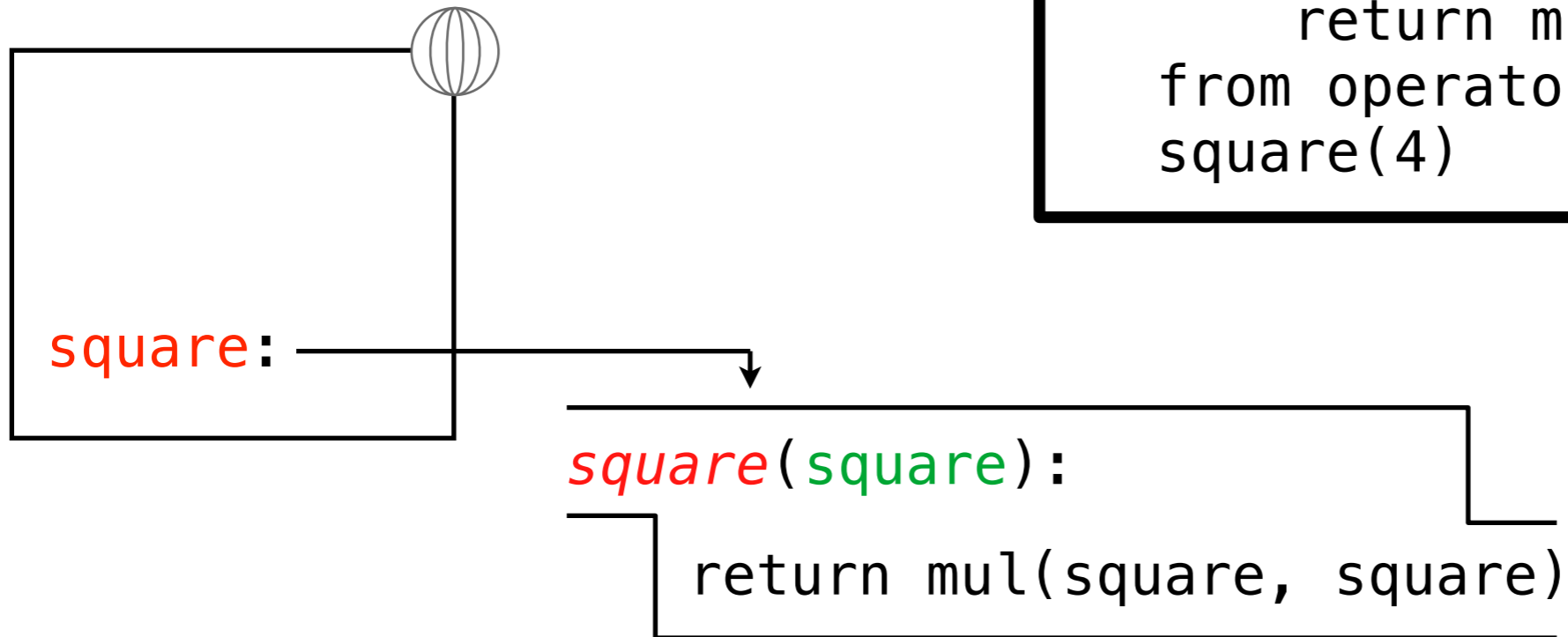


```
▶ def square(square):  
    return mul(square, square)  
from operator import mul  
square(4)
```

# What Happened with `def square(square)`?

---

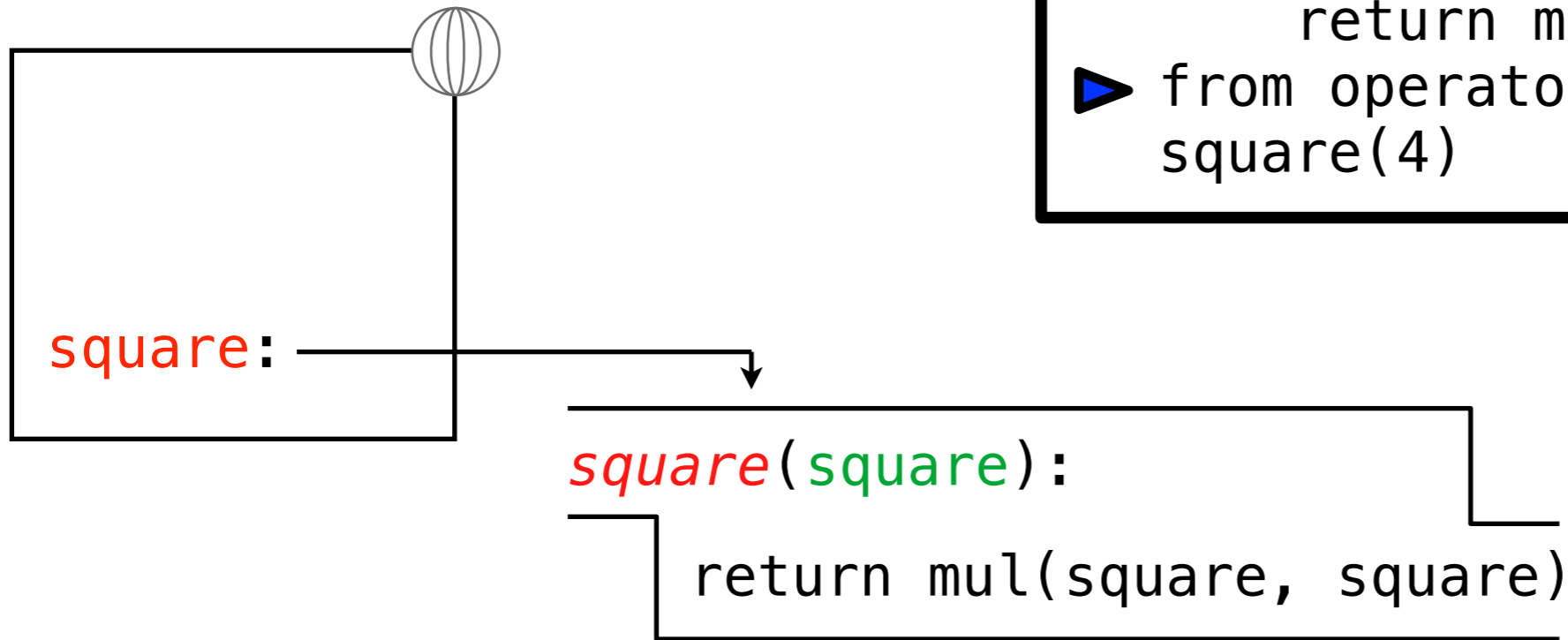
```
▶ def square(square):  
    return mul(square, square)  
from operator import mul  
square(4)
```



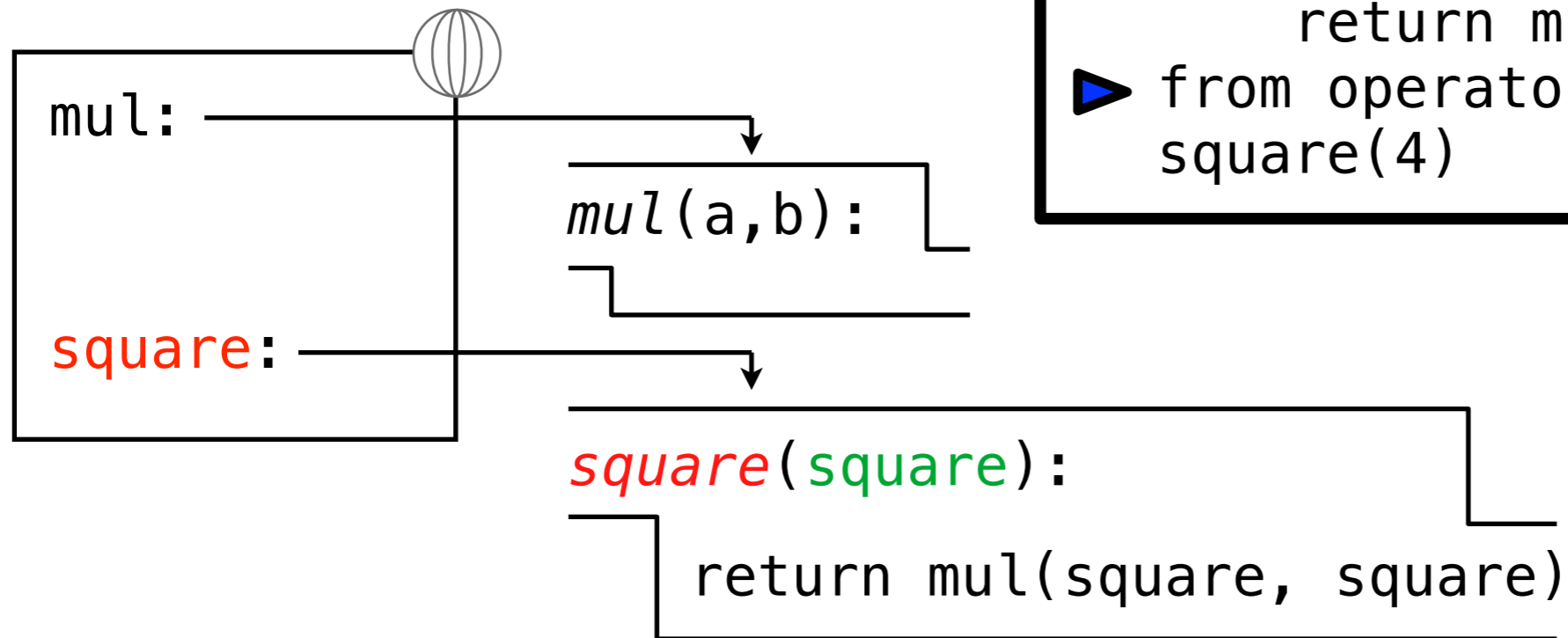
# What Happened with def square(square)?

---

```
def square(square):  
    return mul(square, square)  
▶ from operator import mul  
square(4)
```



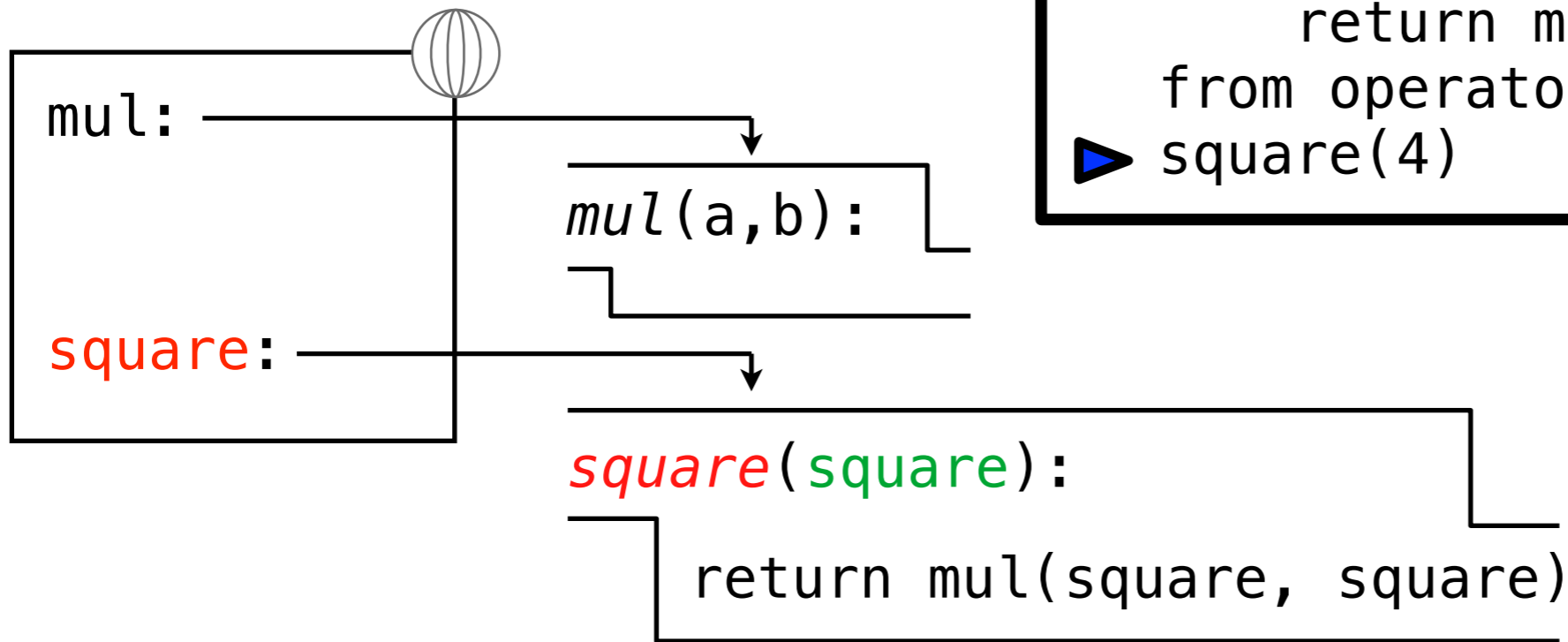
# What Happened with `def square(square)`?



```
def square(square):  
    return mul(square, square)  
▶ from operator import mul  
square(4)
```

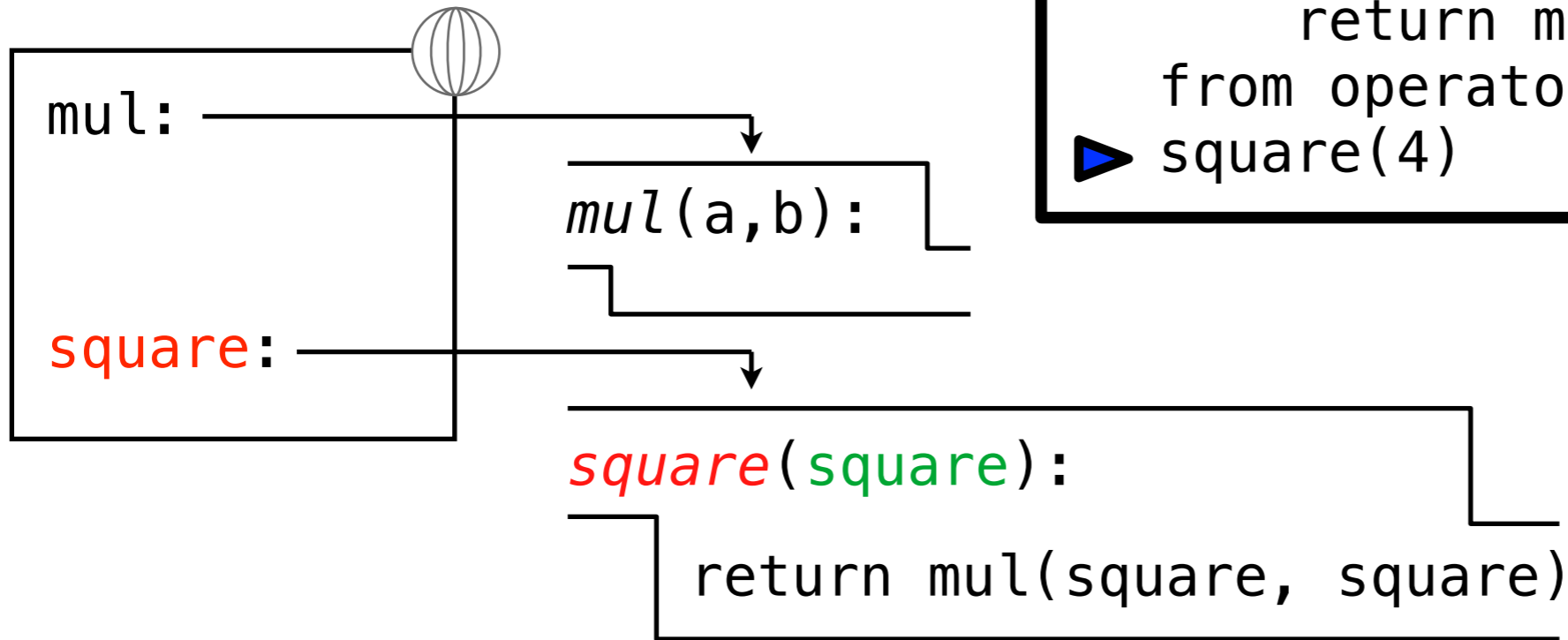
# What Happened with `def square(square)`?

```
def square(square):  
    return mul(square, square)  
from operator import mul  
▶ square(4)
```



# What Happened with `def square(square)`?

```
def square(square):  
    return mul(square, square)  
from operator import mul  
▶ square(4)
```

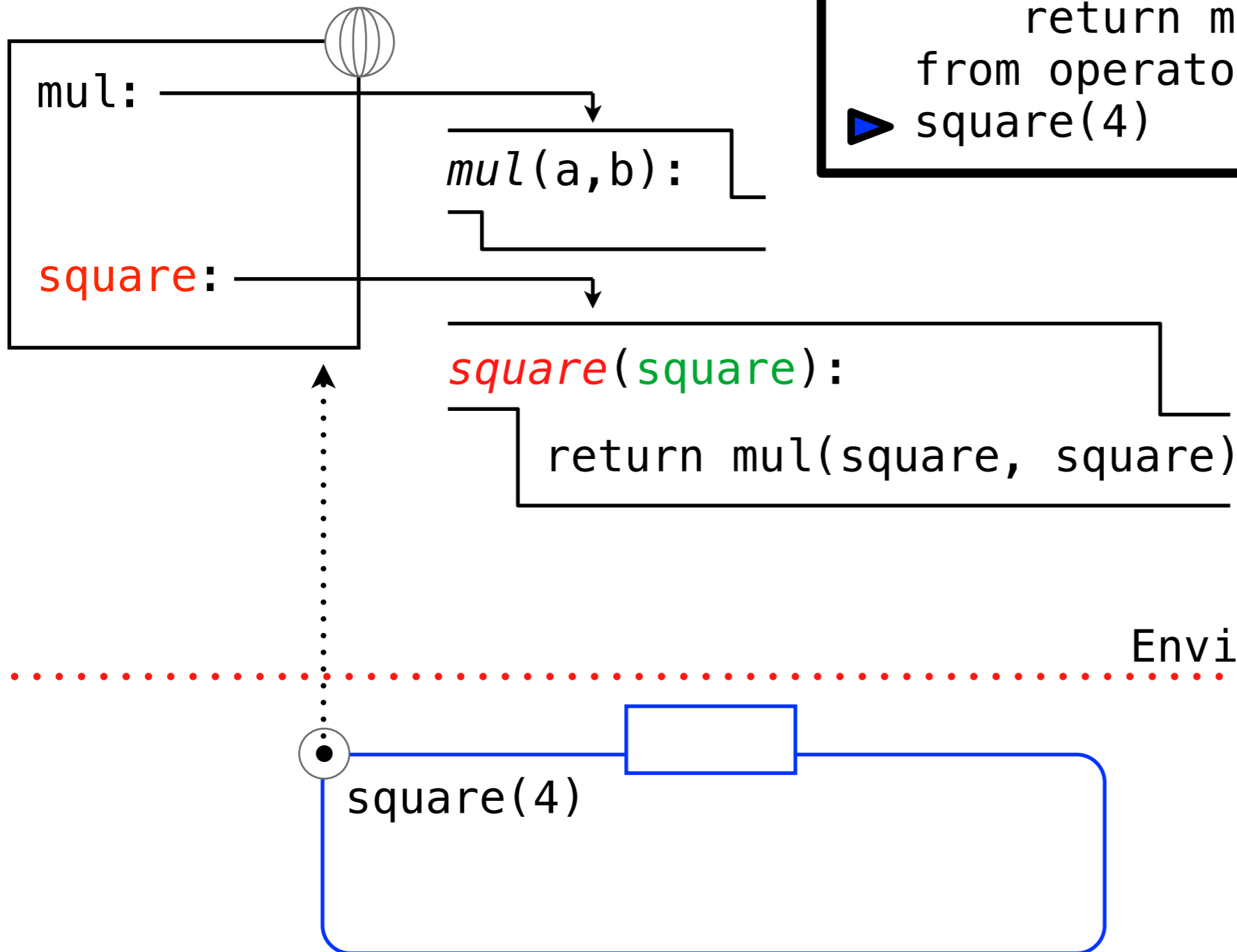


Environments & values  
Expressions



# What Happened with def square(square)?

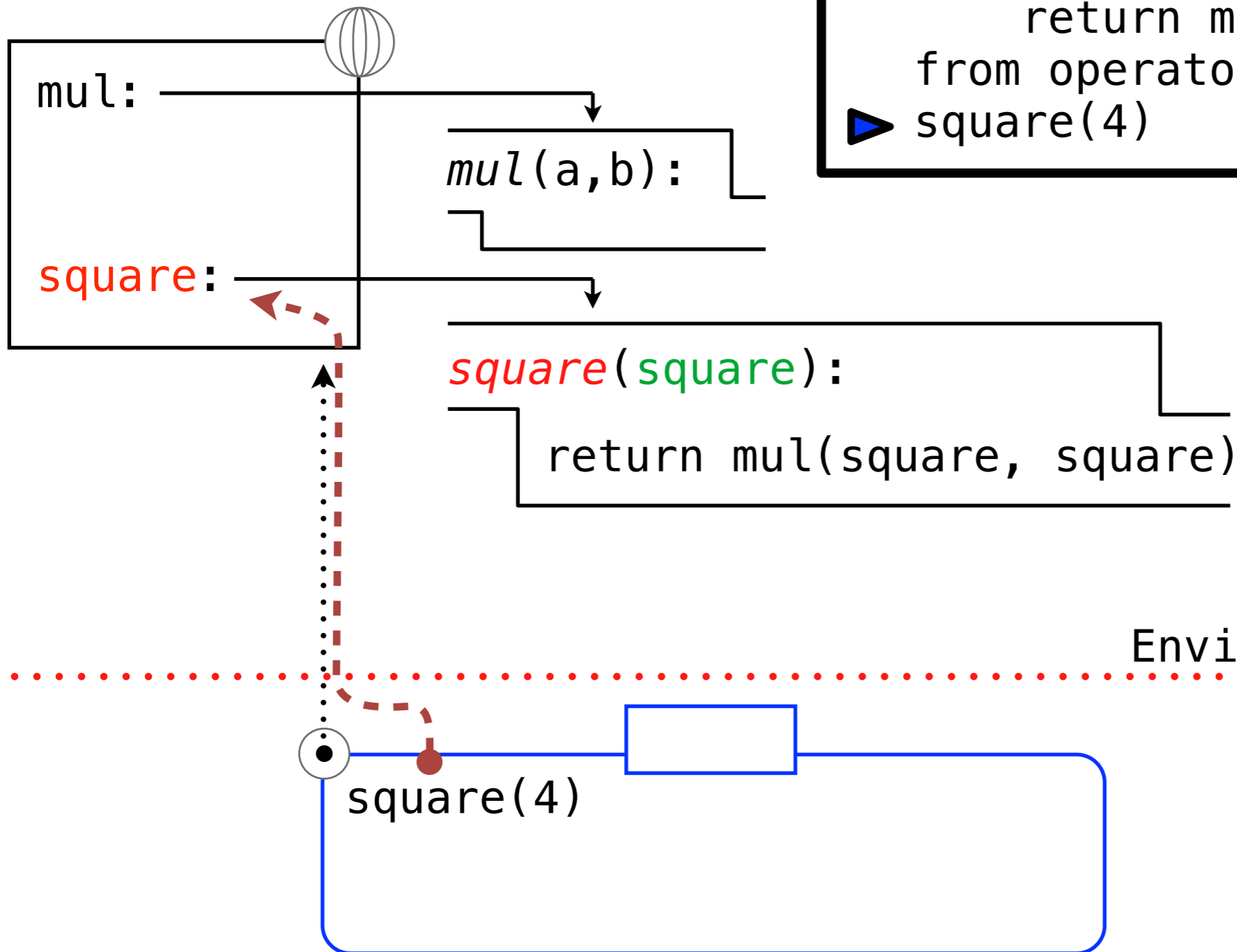
```
def square(square):  
    return mul(square, square)  
from operator import mul  
▶ square(4)
```



Environments & values  
Expressions

# What Happened with def square(square)?

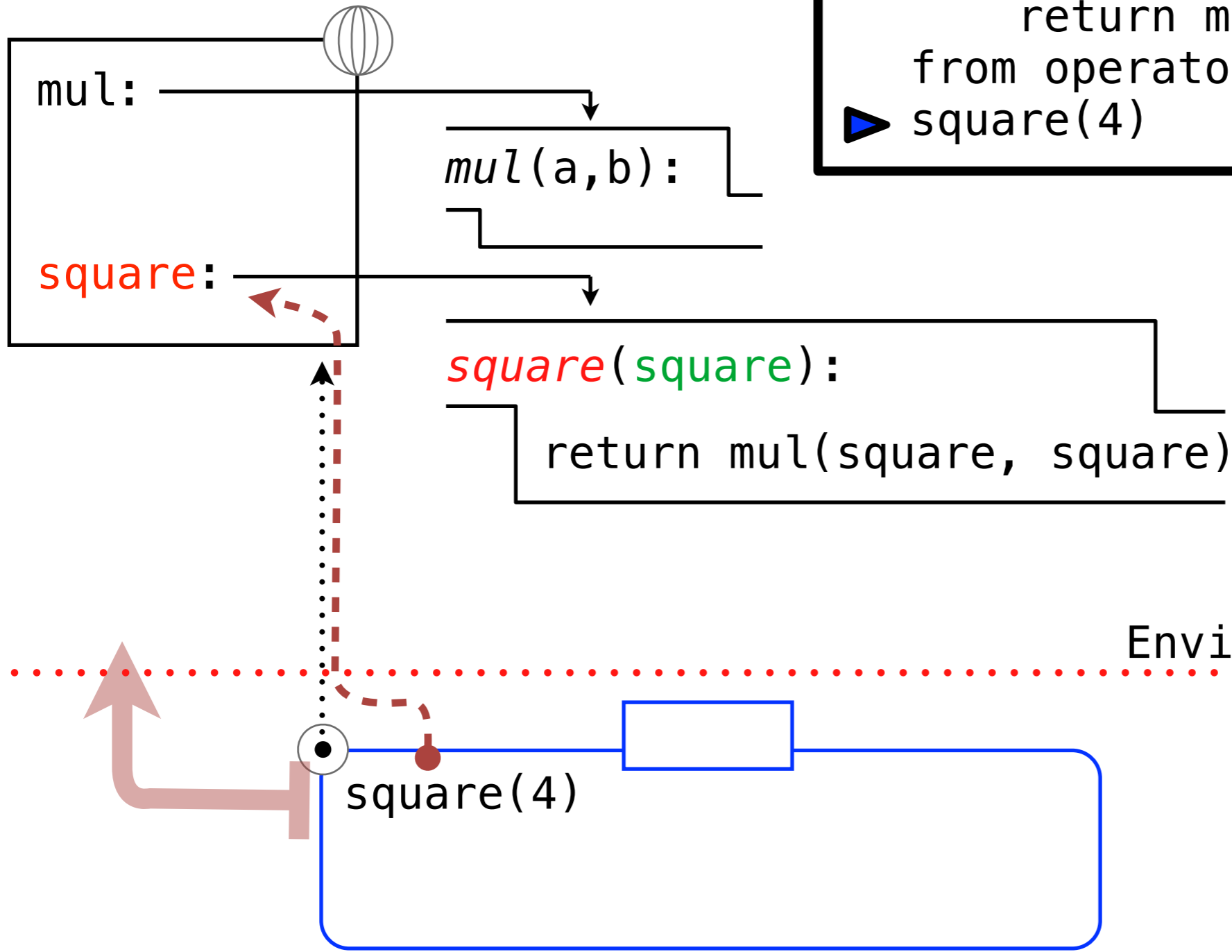
```
def square(square):  
    return mul(square, square)  
from operator import mul  
▶ square(4)
```



Environments & values  
Expressions

# What Happened with def square(square)?

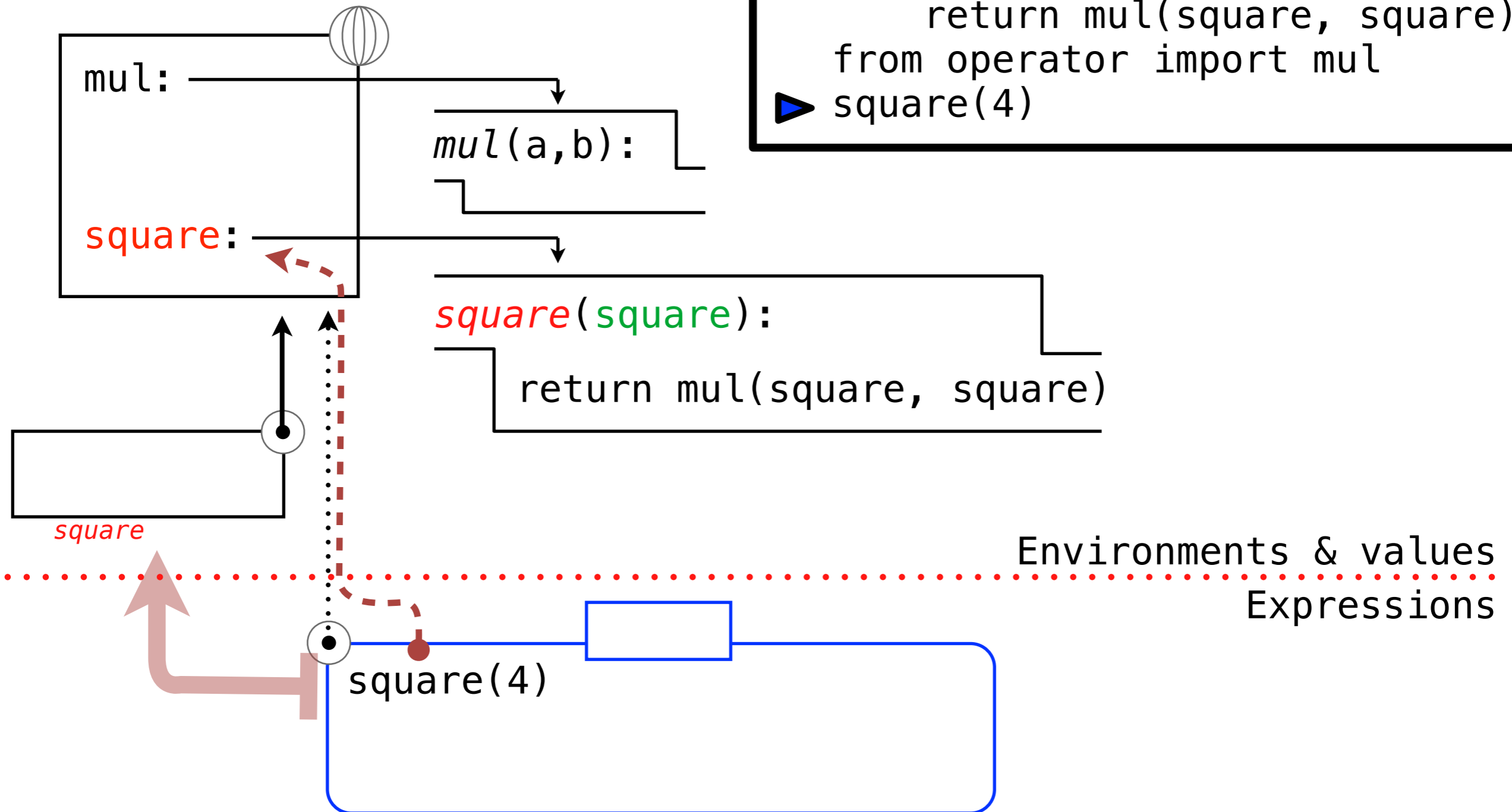
```
def square(square):  
    return mul(square, square)  
from operator import mul  
▶ square(4)
```



Environments & values  
Expressions

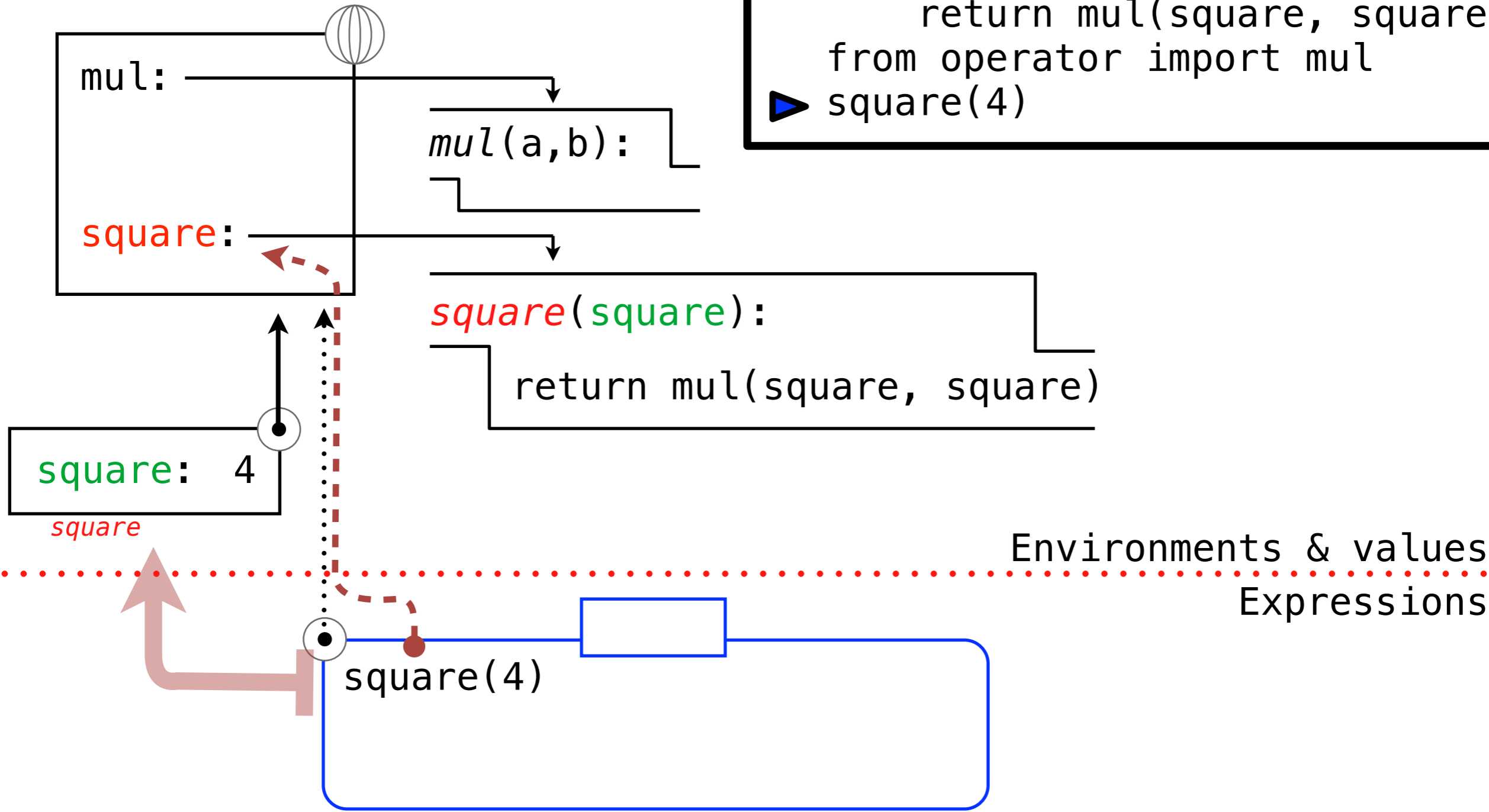
# What Happened with def square(square)?

```
def square(square):  
    return mul(square, square)  
from operator import mul  
▶ square(4)
```



# What Happened with def square(square)?

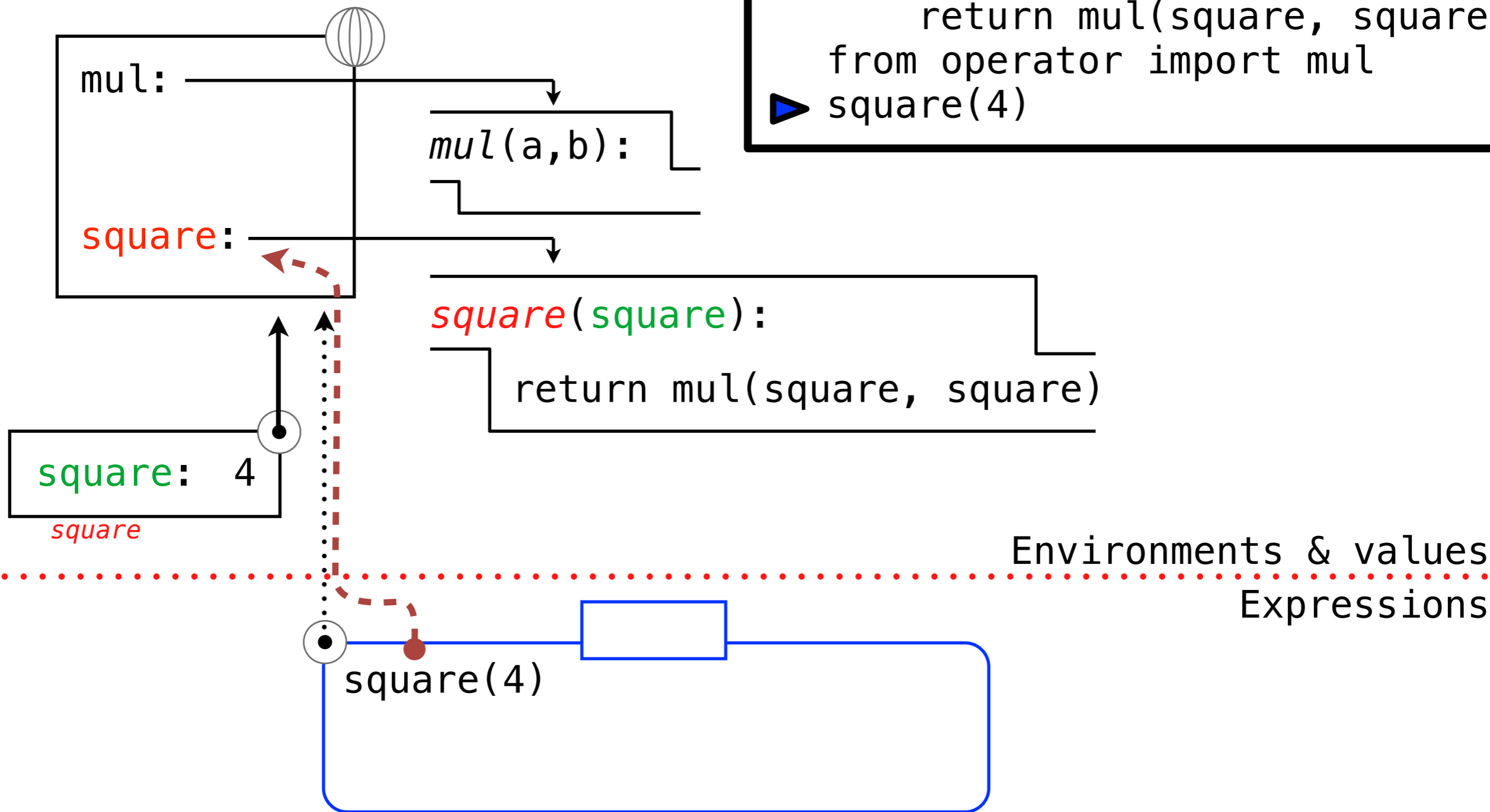
```
def square(square):  
    return mul(square, square)  
from operator import mul  
▶ square(4)
```



Environments & values  
Expressions

# What Happened with def square(square)?

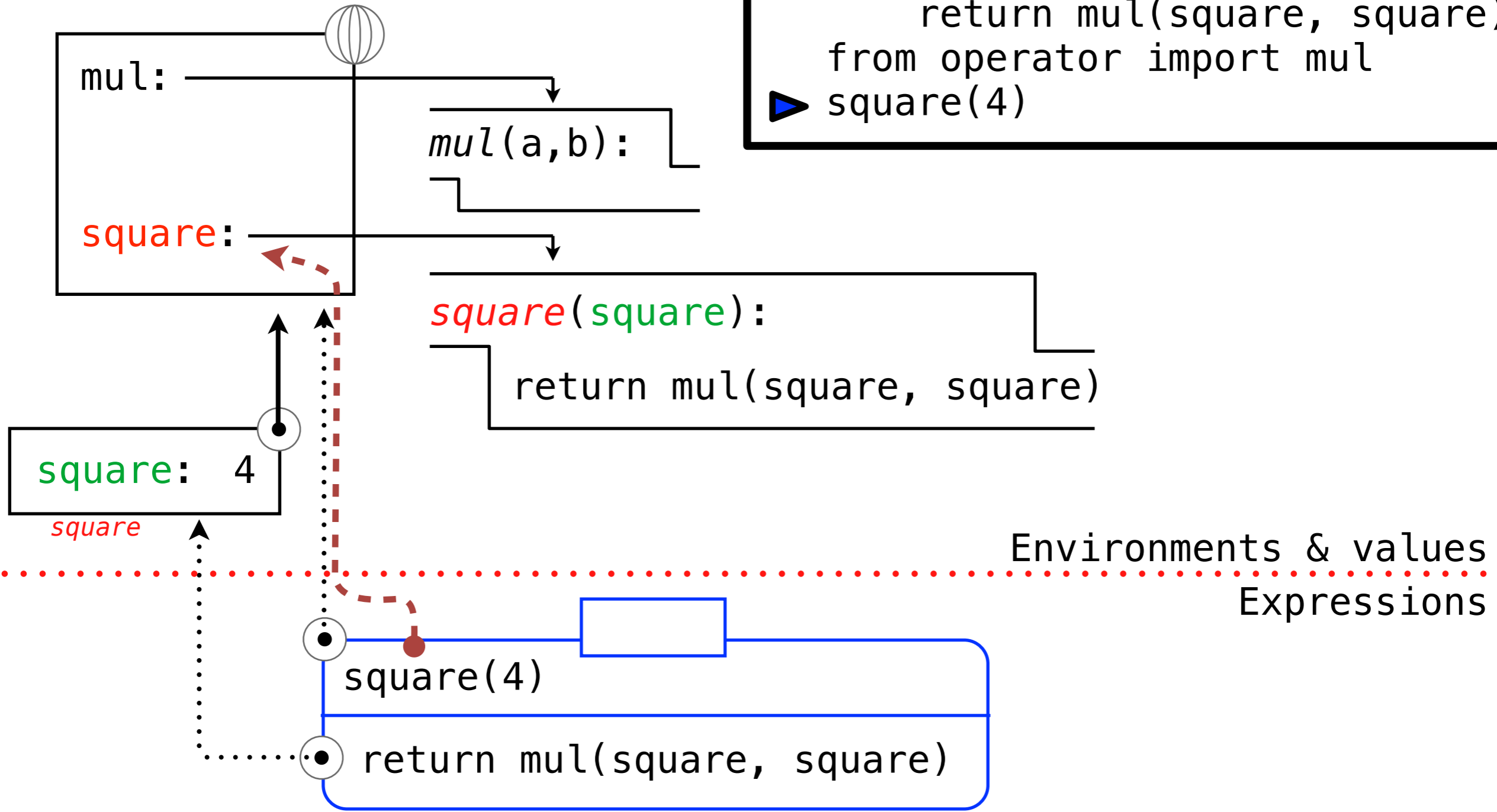
```
def square(square):  
    return mul(square, square)  
from operator import mul  
▶ square(4)
```



Environments & values  
Expressions

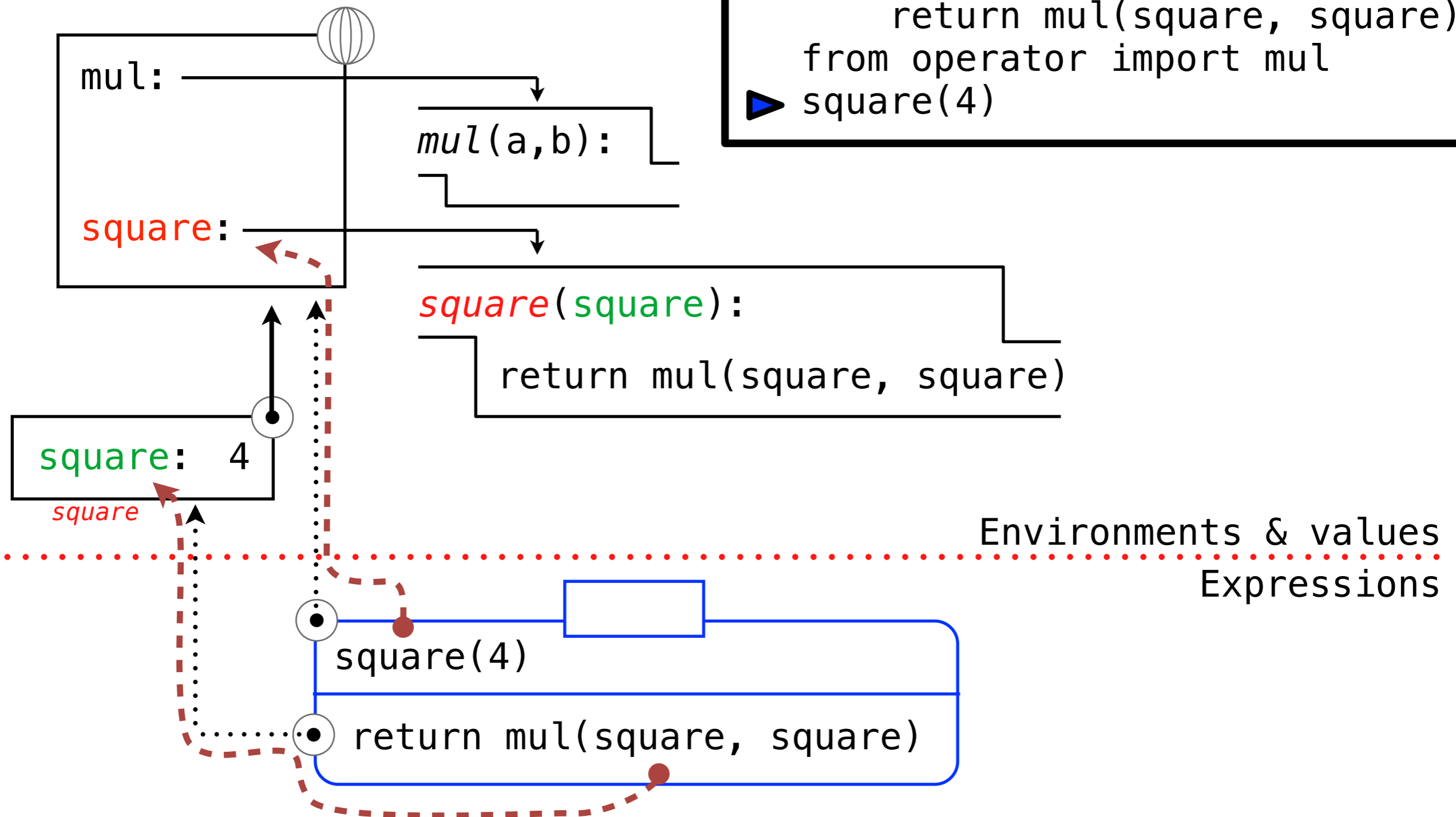
# What Happened with def square(square)?

```
def square(square):  
    return mul(square, square)  
from operator import mul  
▶ square(4)
```



# What Happened with def square(square)?

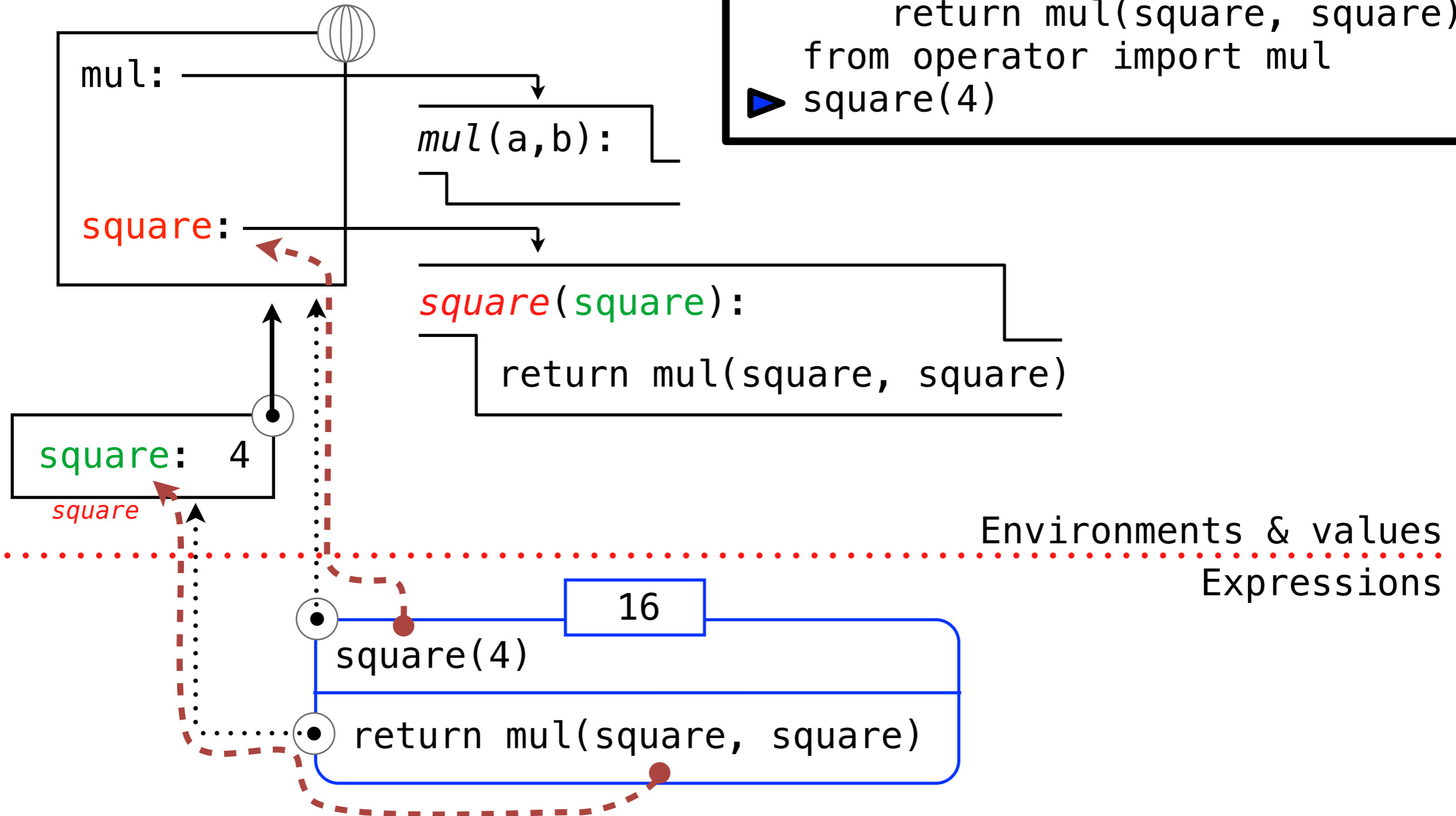
```
def square(square):  
    return mul(square, square)  
from operator import mul  
▶ square(4)
```





# What Happened with def square(square)?

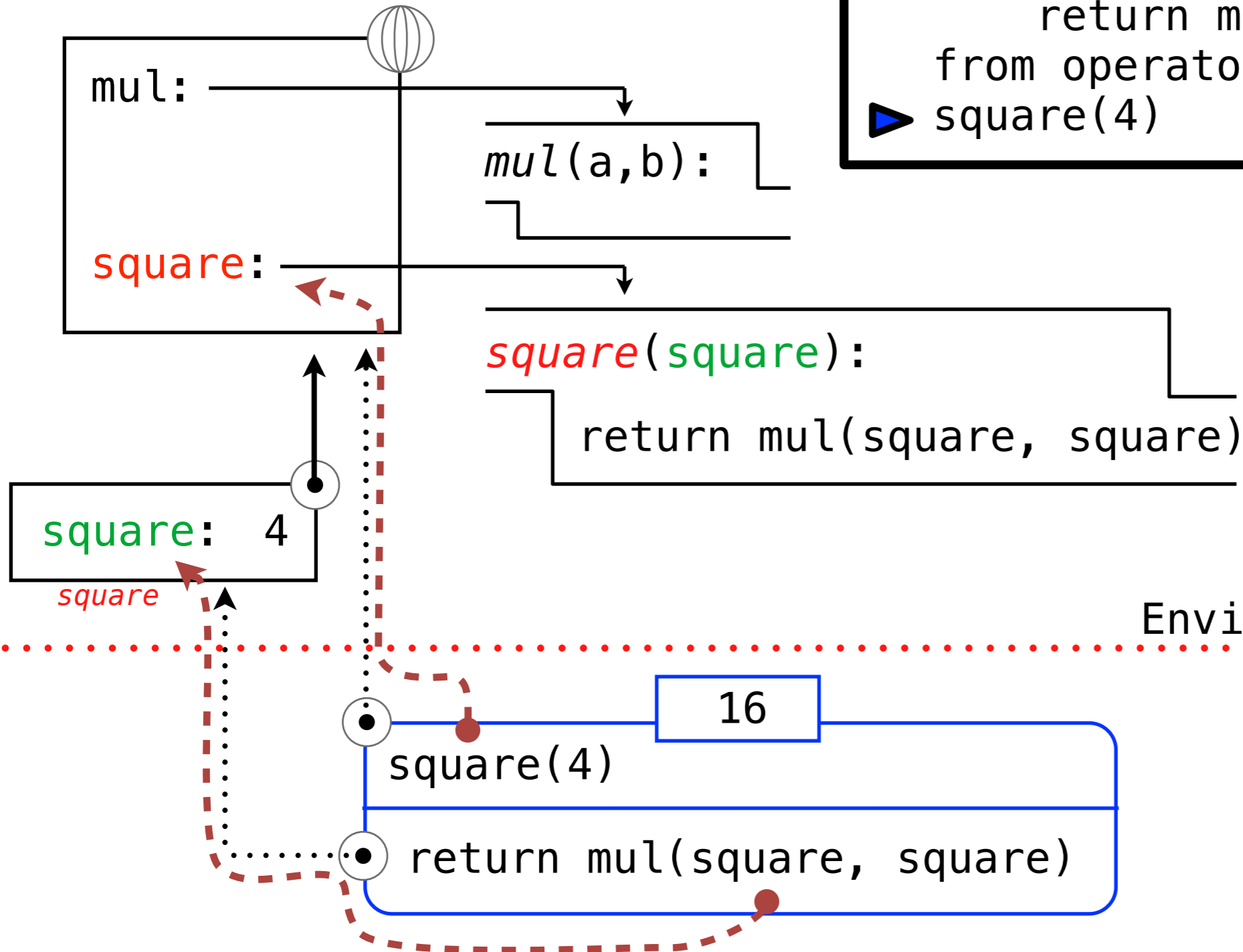
```
def square(square):
    return mul(square, square)
from operator import mul
▶ square(4)
```



# What Happened with def square(square)?

```
def square(square):  
    return mul(square, square)  
from operator import mul  
▶ square(4)
```

Don't actually name functions and formal parameters the same thing!

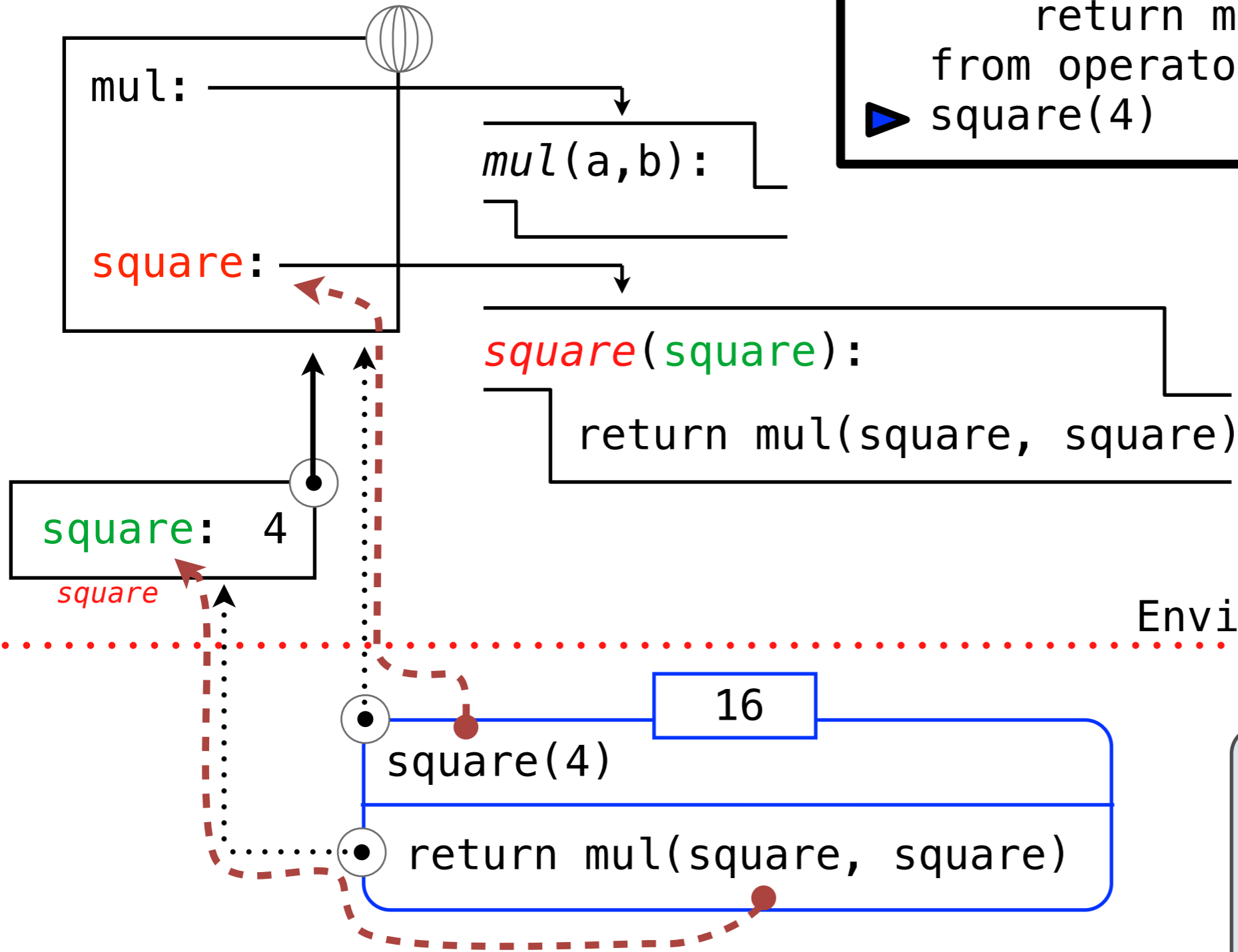


Environments & values  
Expressions

# What Happened with def square(square)?

```
def square(square):  
    return mul(square, square)  
from operator import mul  
▶ square(4)
```

Don't actually name functions and formal parameters the same thing!



Environments & values  
Expressions

Environment diagrams will be on holiday until Wednesday

# Statements

---

A statement  
is executed by the interpreter  
to perform an action

# Statements

---

A statement  
is executed by the interpreter  
to perform an action

## Compound statements:

```
<header>:  
    <statement>  
    <statement>  
    ...  
<separating header>:  
    <statement>  
    <statement>  
    ...  
...
```

# Statements

---

A statement  
is executed by the interpreter  
to perform an action

## Compound statements:

Statement

```
<header>:  
    <statement>  
    <statement>  
    ...  
<separating header>:  
    <statement>  
    <statement>  
    ...  
...
```

# Statements

---

A statement  
is executed by the interpreter  
to perform an action

## Compound statements:

Statement

Clause

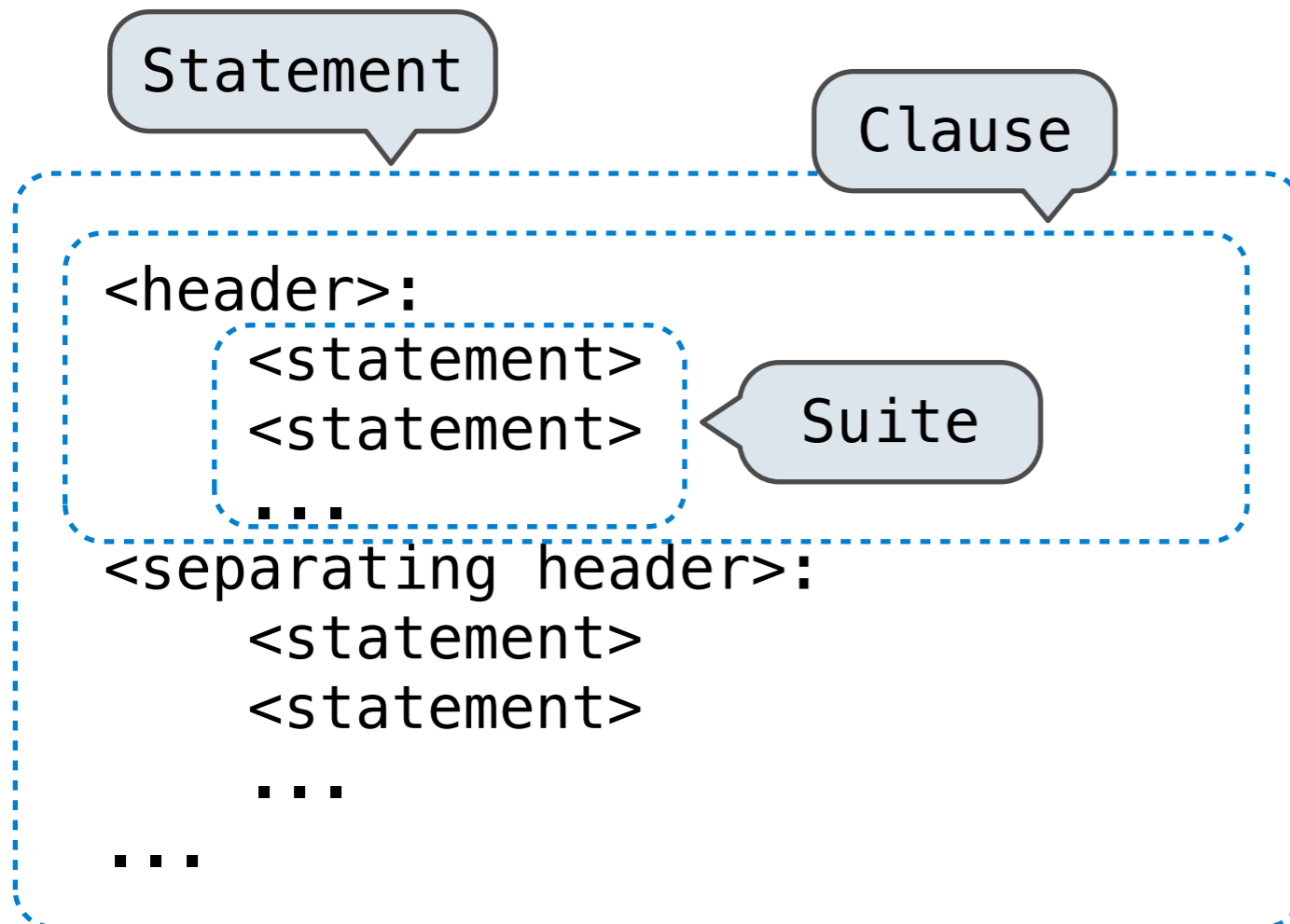
```
<header>:  
  <statement>  
  <statement>  
  ...  
<separating header>:  
  <statement>  
  <statement>  
  ...  
...
```

# Statements

---

A statement  
is executed by the interpreter  
to perform an action

## Compound statements:



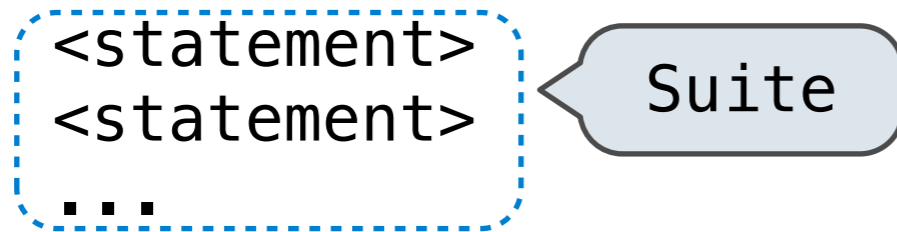


# Compound Statements

---

## Compound statements:

<header>:



<separating header>:

`<statement>`  
`<statement>`

...

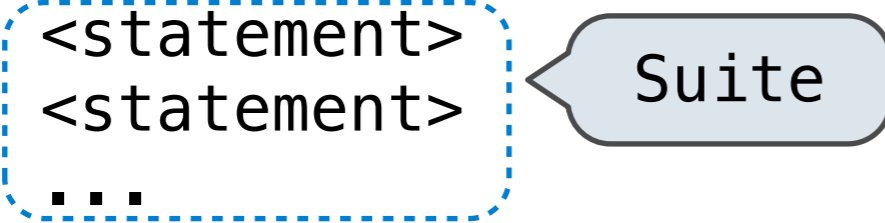
...

# Compound Statements

---

## Compound statements:

```
<header>:  
  <statement>  
  <statement>  
  ...  
<separating header>:  
  <statement>  
  <statement>  
  ...  
...
```



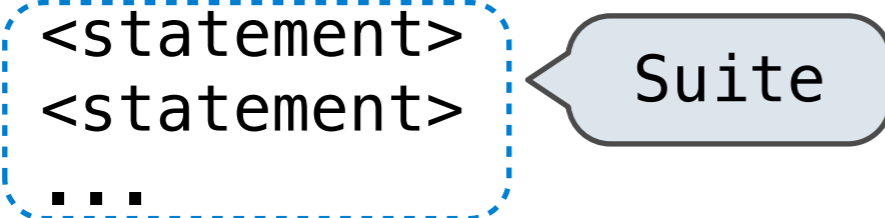
A suite is a sequence  
of statements

# Compound Statements

---

## Compound statements:

```
<header>:  
  <statement>  
  <statement>  
  ...  
<separating header>:  
  <statement>  
  <statement>  
  ...  
...
```



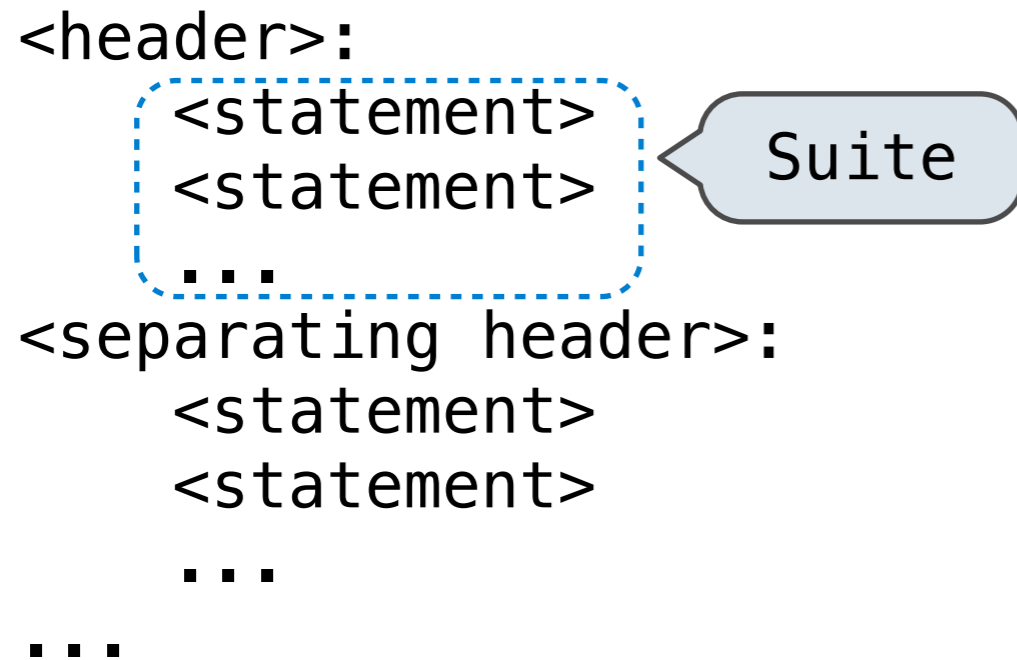
A suite is a sequence of statements

To “execute” a suite means to execute its sequence of statements, in order

# Compound Statements

---

## Compound statements:



A suite is a sequence of statements

To “execute” a suite means to execute its sequence of statements, in order

## Execution Rule for a sequence of statements:

- Execute the first
- Unless directed otherwise, execute the rest

# The Fibonacci Sequence

---

# The Fibonacci Sequence

---

$0, 1, 1, 2, 3, 5, 8, 13, \dots$

# The Fibonacci Sequence

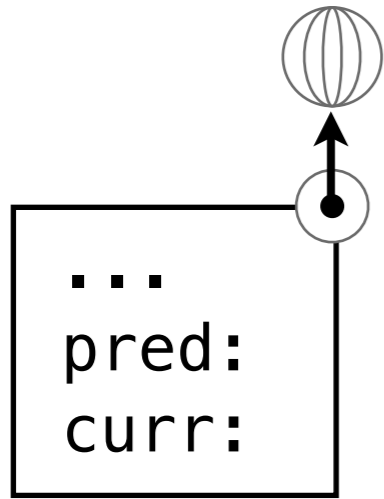
---

0, 1, 1, 2, 3, 5, 8, 13, ...

```
def fib(n):  
    """Compute the nth Fibonacci number, for n >= 2."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 2                # Tracks which Fib number is curr  
    while k < n:  
        pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

# The Fibonacci Sequence

---



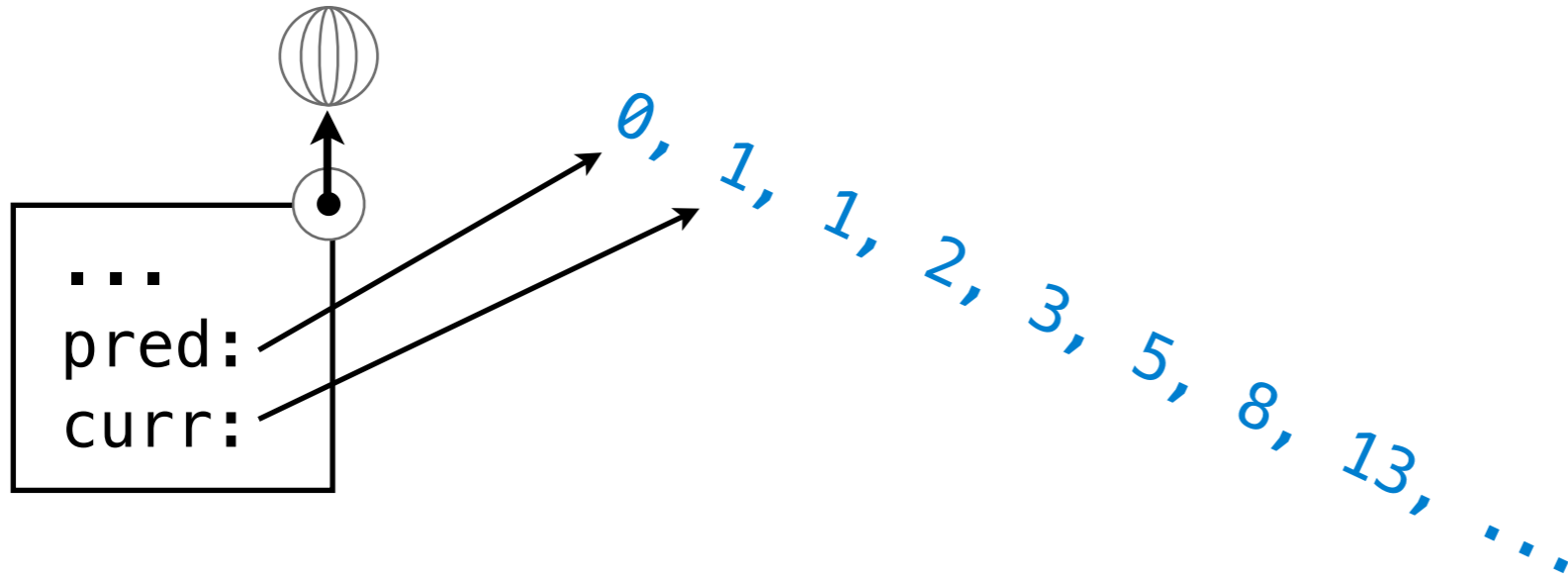
0, 1, 1, 2, 3, 5, 8, 13, ...

```
def fib(n):  
    """Compute the nth Fibonacci number, for n >= 2."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 2                # Tracks which Fib number is curr  
    while k < n:  
        pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```



# The Fibonacci Sequence

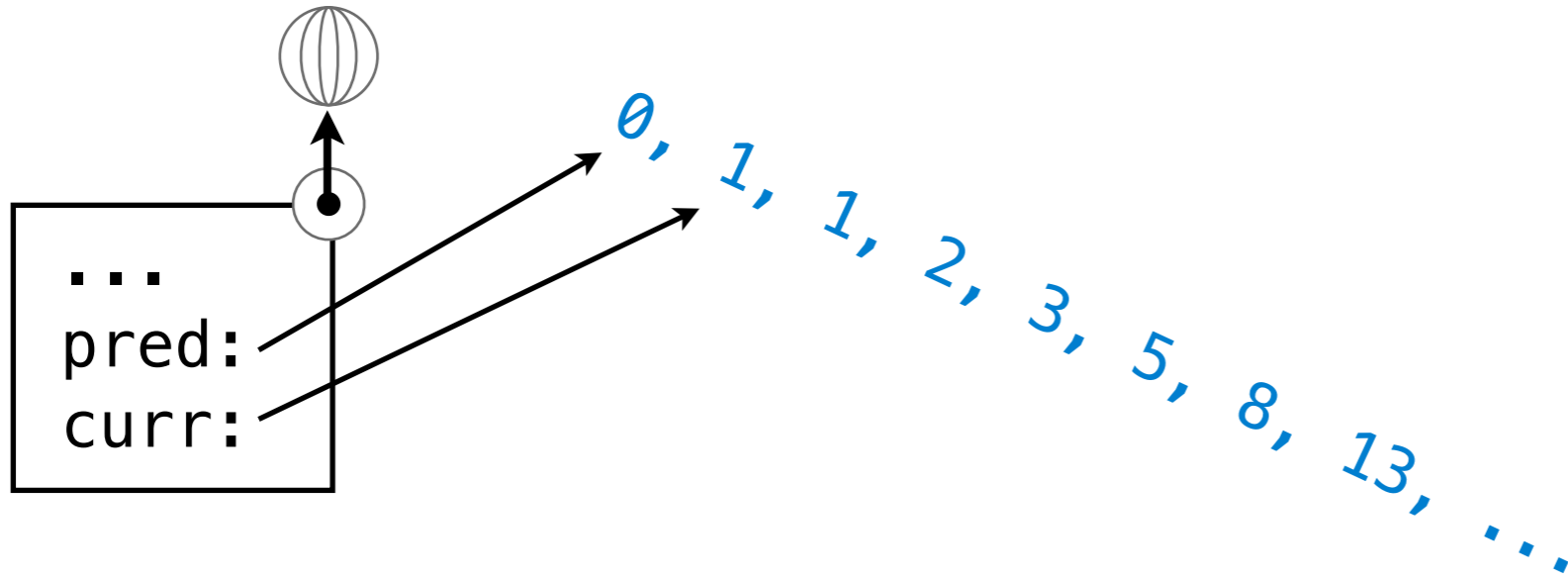
---



```
def fib(n):  
    """Compute the nth Fibonacci number, for n >= 2."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 2                # Tracks which Fib number is curr  
    while k < n:  
        pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

# The Fibonacci Sequence

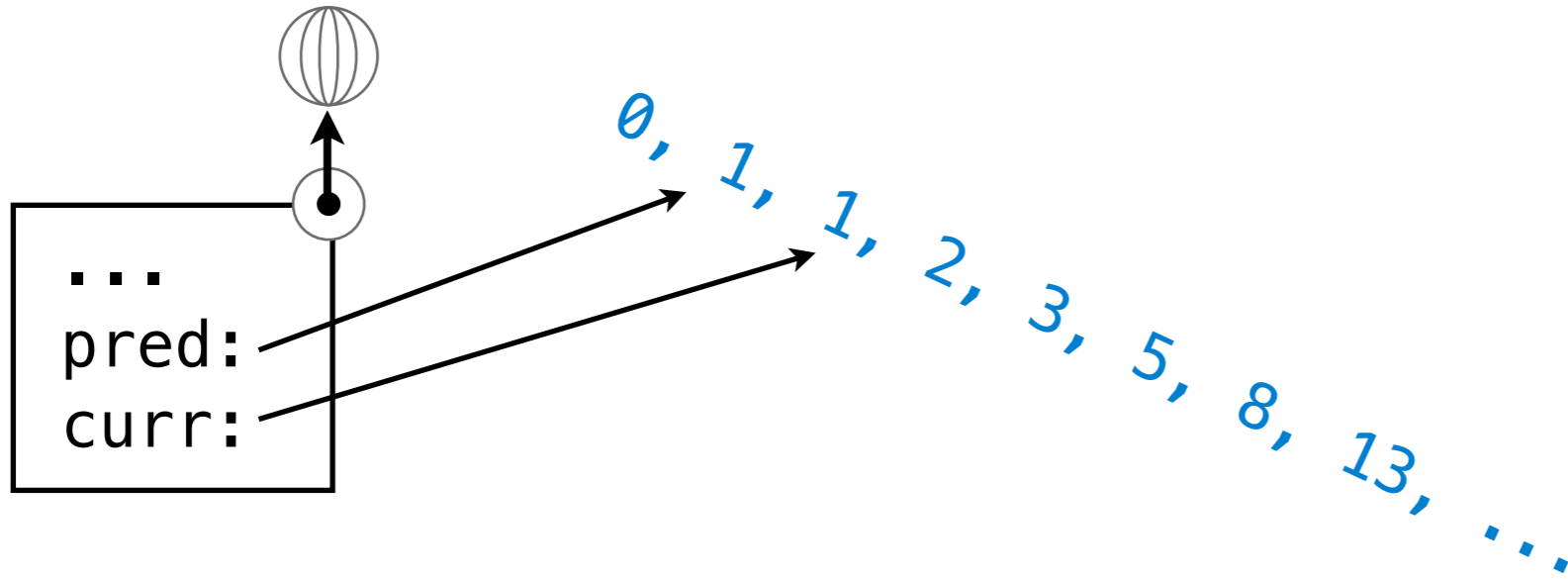
---



```
def fib(n):  
    """Compute the nth Fibonacci number, for n >= 2."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 2                # Tracks which Fib number is curr  
    while k < n:  
        ▶ pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

# The Fibonacci Sequence

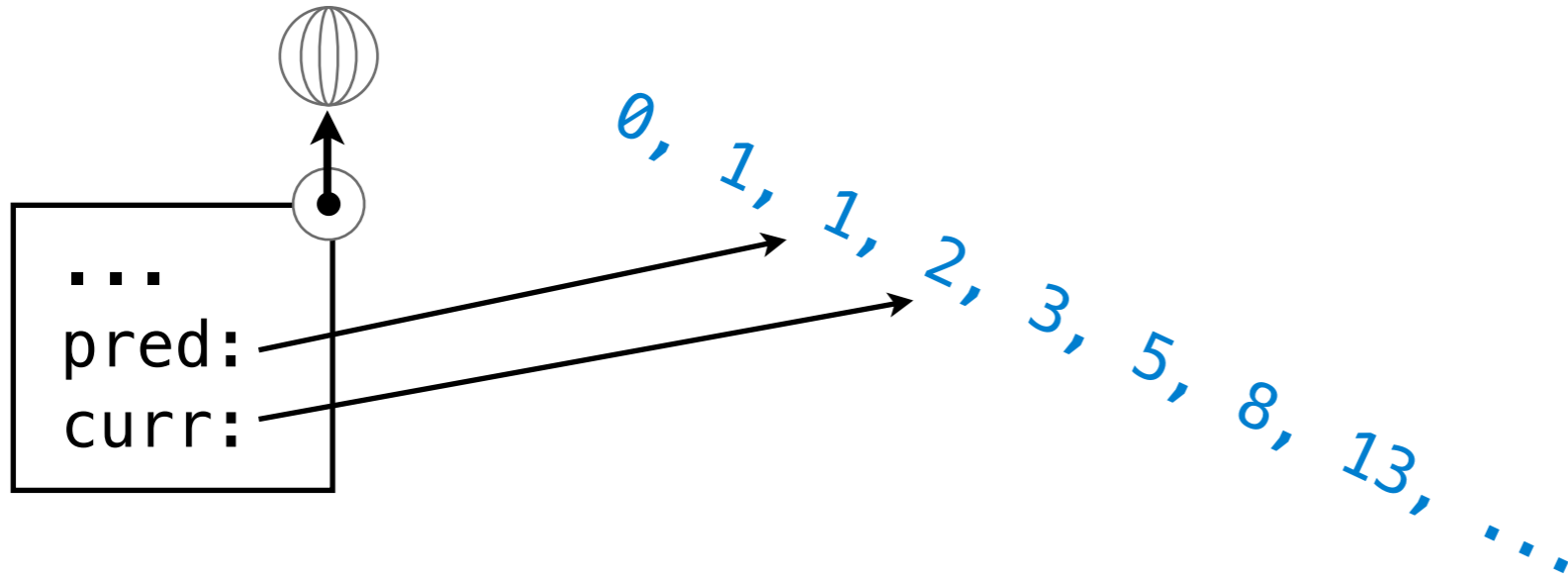
---



```
def fib(n):  
    """Compute the nth Fibonacci number, for n >= 2."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 2                # Tracks which Fib number is curr  
    while k < n:  
        ▶ pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

# The Fibonacci Sequence

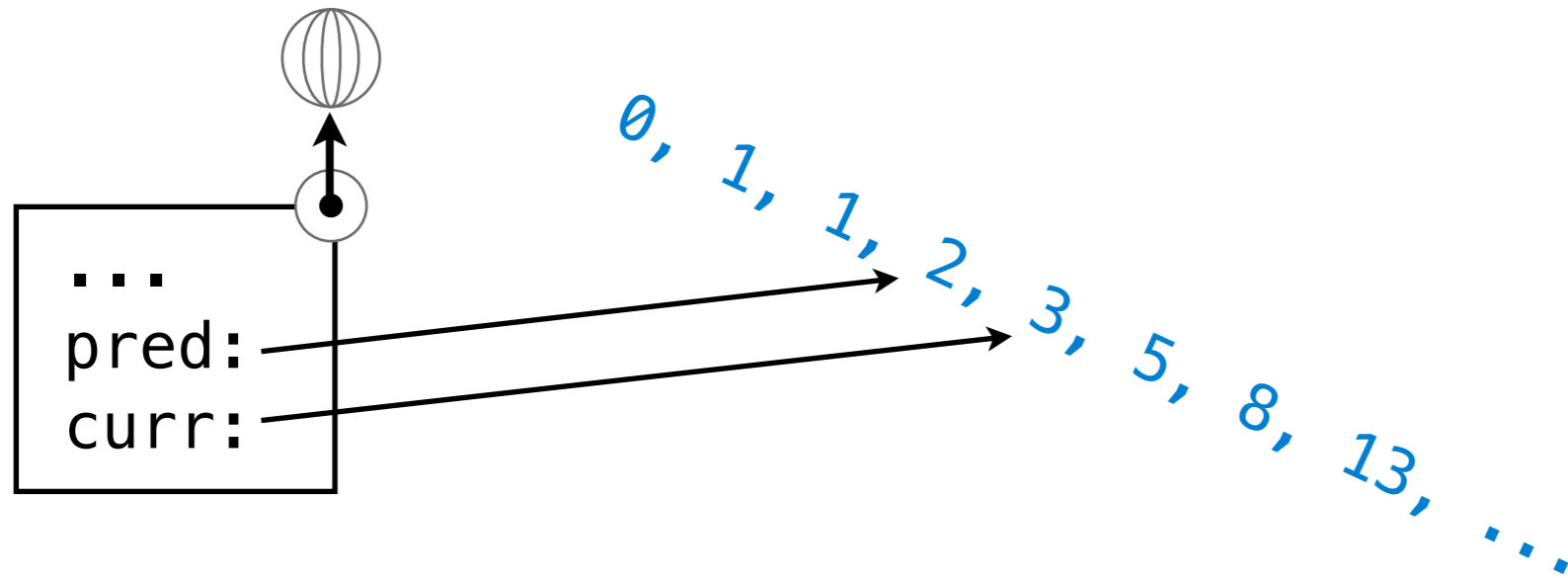
---



```
def fib(n):  
    """Compute the nth Fibonacci number, for n >= 2."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 2                # Tracks which Fib number is curr  
    while k < n:  
        ▶ pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

# The Fibonacci Sequence

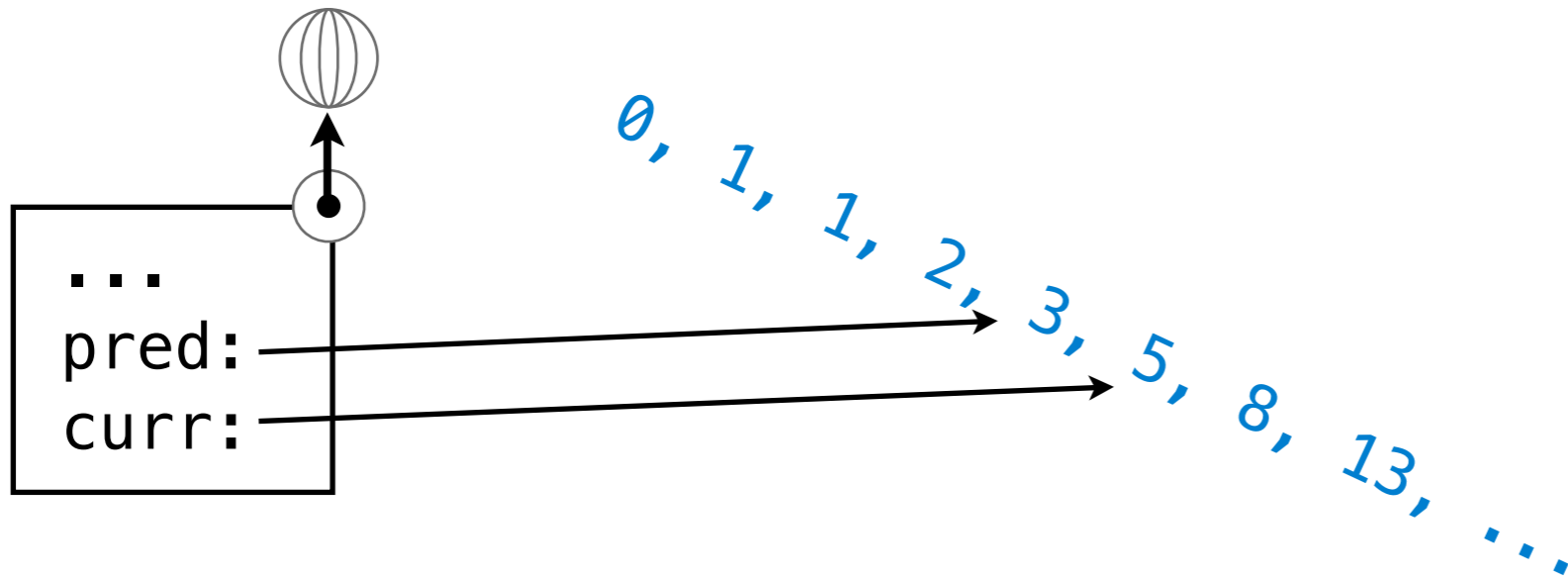
---



```
def fib(n):  
    """Compute the nth Fibonacci number, for n >= 2."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 2                # Tracks which Fib number is curr  
    while k < n:  
        ▶ pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

# The Fibonacci Sequence

---



```
def fib(n):  
    """Compute the nth Fibonacci number, for n >= 2."""  
    pred, curr = 0, 1    # First two Fibonacci numbers  
    k = 2                # Tracks which Fib number is curr  
    while k < n:  
        ▶ pred, curr = curr, pred + curr  
        k = k + 1  
    return curr
```

# Higher-Order Functions Introduction

---

(Demo)

# Pig Introduction

---

( Demo )