61A Lecture 3

Wednesday, August 31

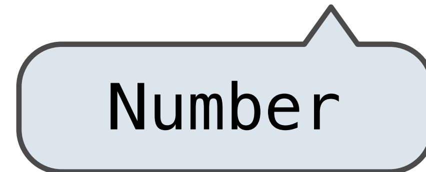
Lightning Review: Expressions

Primitive expressions:

Call expressions:

Lightning Review: Expressions

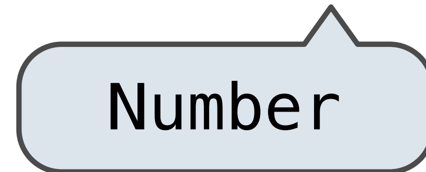
Primitive expressions: 2



Call expressions:

Lightning Review: Expressions

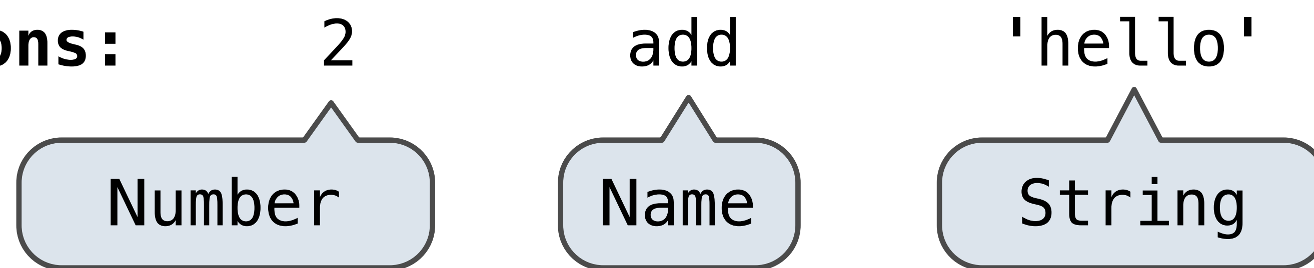
Primitive expressions: 2 add



Call expressions:

Lightning Review: Expressions

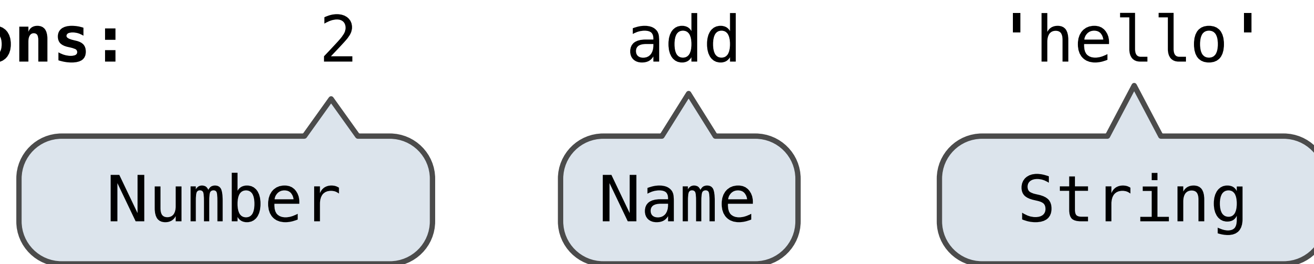
Primitive expressions:



Call expressions:

Lightning Review: Expressions

Primitive expressions:

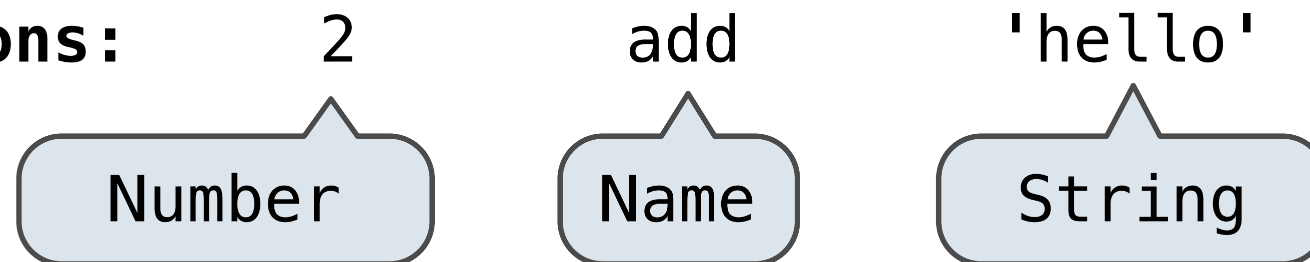


Call expressions:

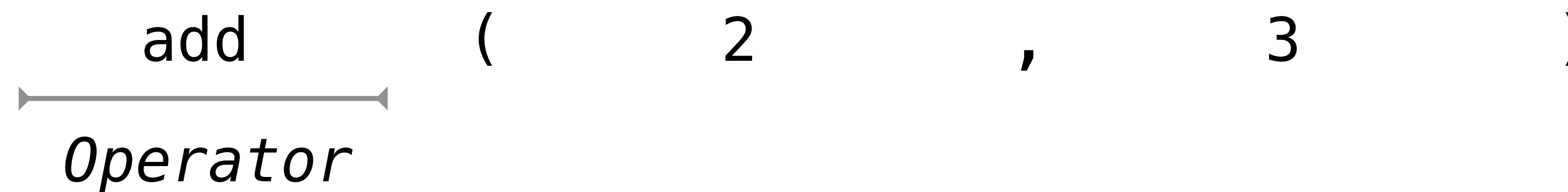
add (2 , 3)

Lightning Review: Expressions

Primitive expressions:

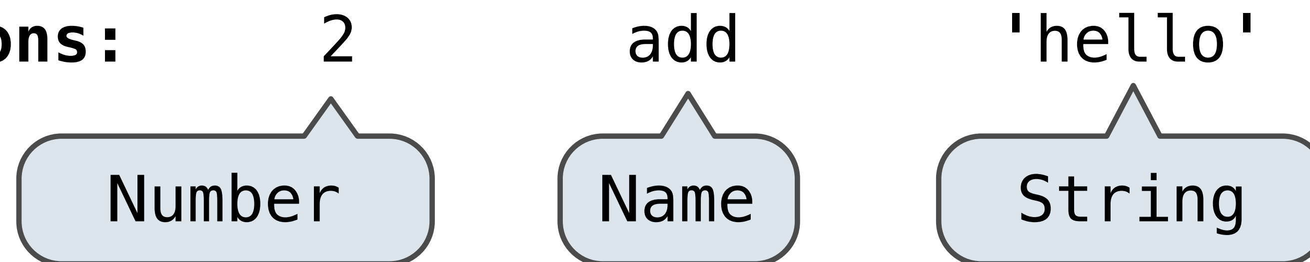


Call expressions:

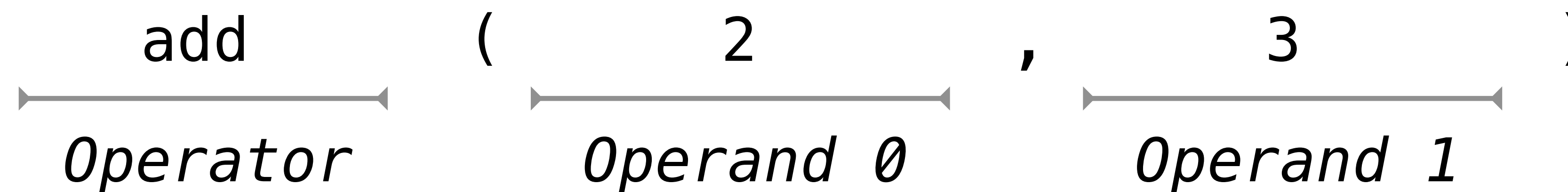


Lightning Review: Expressions

Primitive expressions:

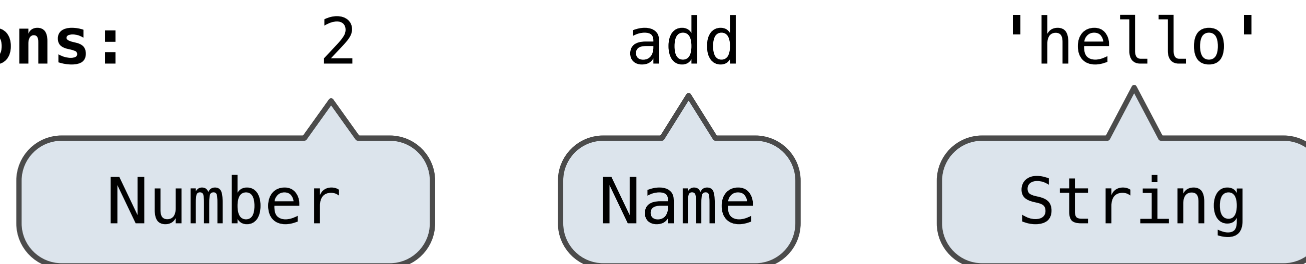


Call expressions:

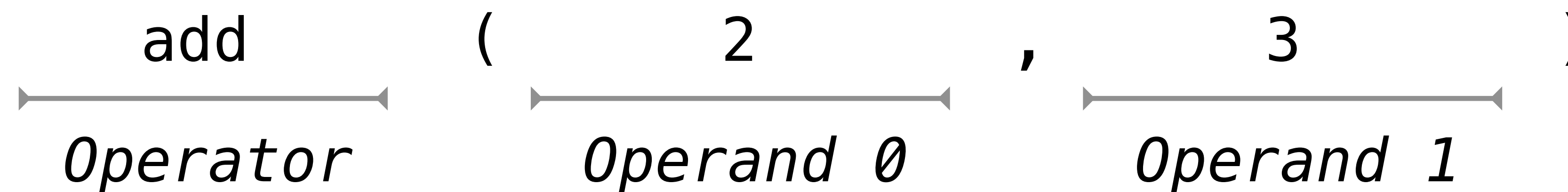


Lightning Review: Expressions

Primitive expressions:



Call expressions:



One big nested call expression

```
mul(add(2, mul(4, 6)), add(3, 5))
```

Life Cycle of a User-Defined Function

What happens?

Defining:

```
>>> def square( x ):
        return mul(x, x)
```

Call expression: `square(2+2)`

Calling/Applying:

```
square( x ):
    return mul(x, x)
```

Life Cycle of a User-Defined Function

What happens?

Defining:

```
>>> def square( x ):
        return mul(x, x)
```

Def statement

Call expression: square(2+2)

Calling/Applying:

```
square( x ):
    return mul(x, x)
```

Life Cycle of a User-Defined Function

What happens?

Defining:

Formal parameter

Def statement

```
>>> def square( x ):
        return mul(x, x)
```

Call expression: square(2+2)

Calling/Applying:

```
square( x ):
    return mul(x, x)
```

Life Cycle of a User-Defined Function

What happens?

Defining:

```
>>> def square( x ):
        return mul(x, x)
```

Formal parameter

Def statement

Body

Call expression: `square(2+2)`

Calling/Applying:

```
square( x ):
    return mul(x, x)
```

Life Cycle of a User-Defined Function

What happens?

Defining:

```
>>> def square( x ):
        return mul(x, x)
```

Formal parameter

Def statement

Body (*return statement*)

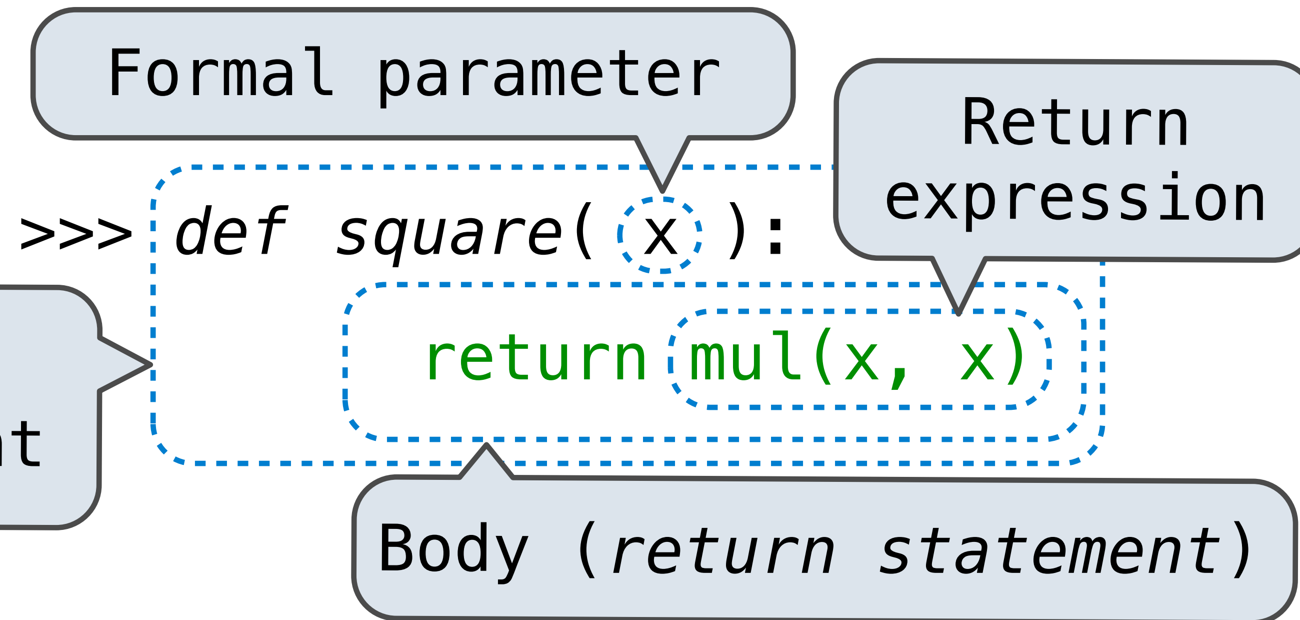
Call expression: `square(2+2)`

Calling/Applying:

```
square( x ):
    return mul(x, x)
```

Life Cycle of a User-Defined Function

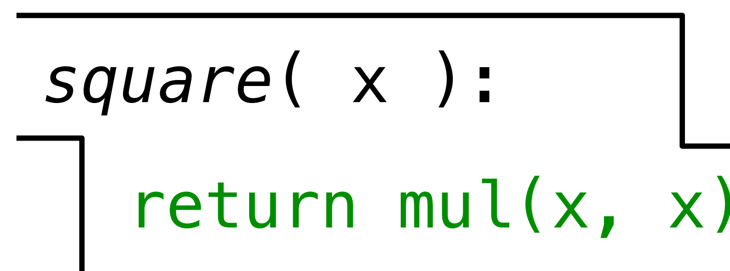
Defining:



What happens?

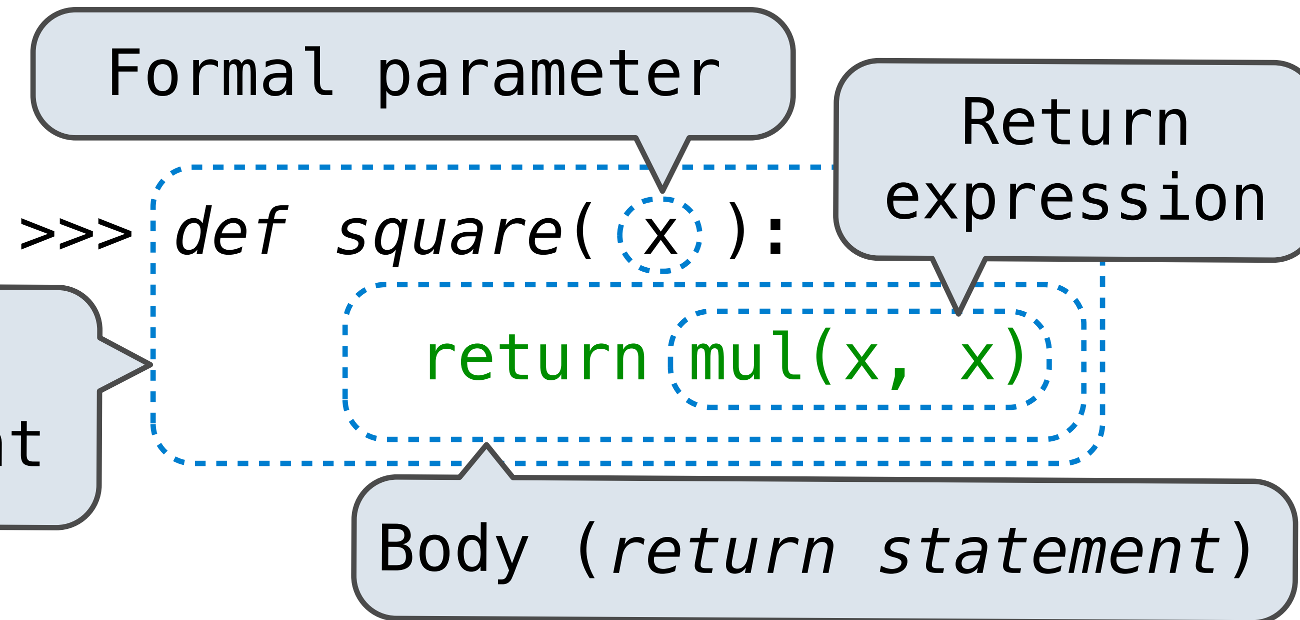
Call expression: square(2+2)

Calling/Applying:



Life Cycle of a User-Defined Function

Defining:

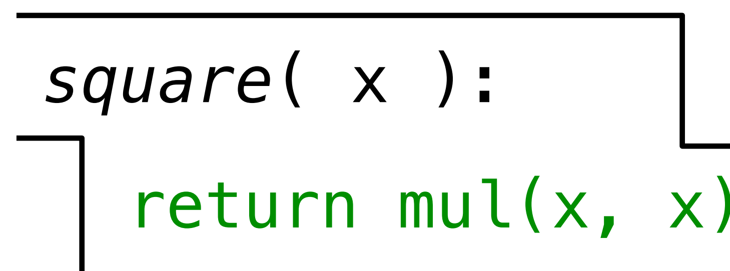


What happens?

Function created

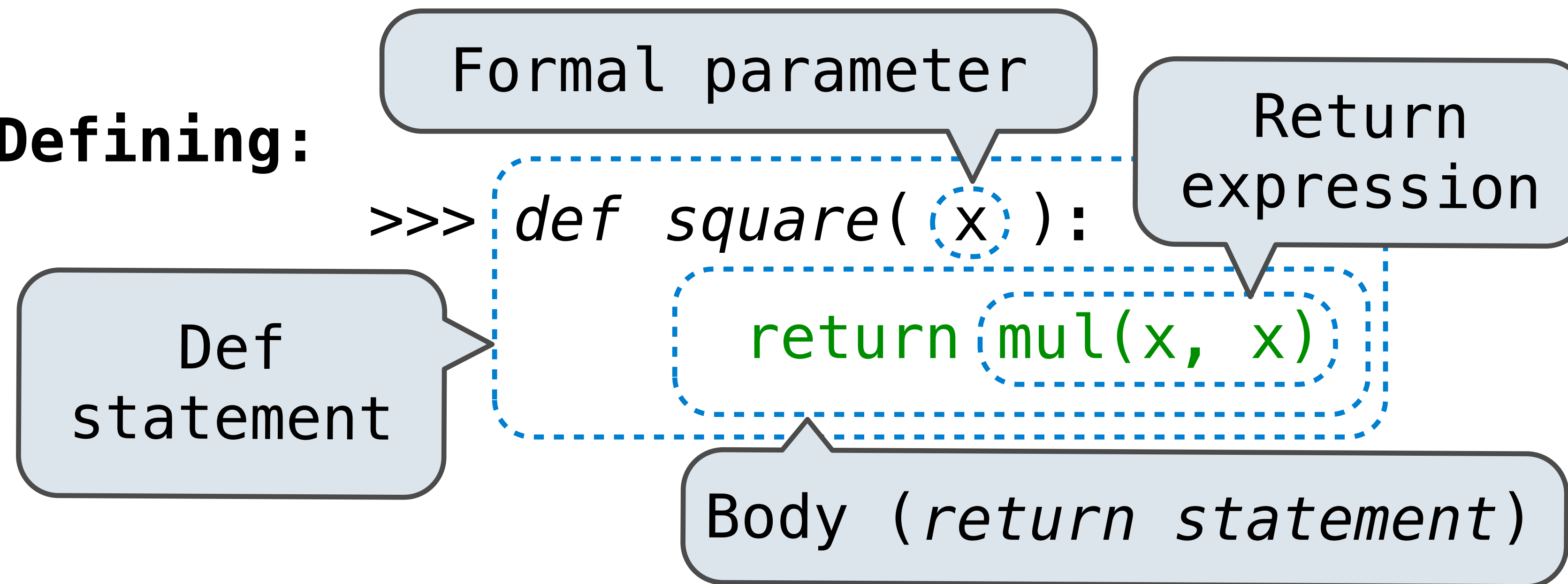
Call expression: square(2+2)

Calling/Applying:



Life Cycle of a User-Defined Function

Defining:



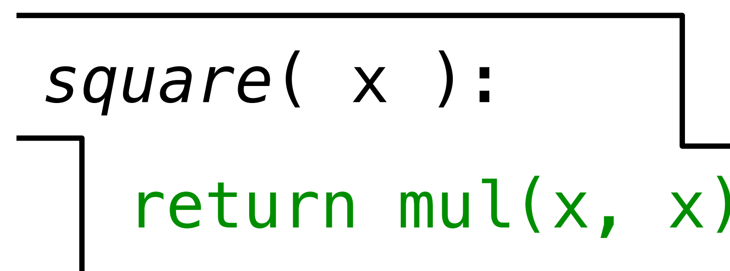
What happens?

Function created

Body stored

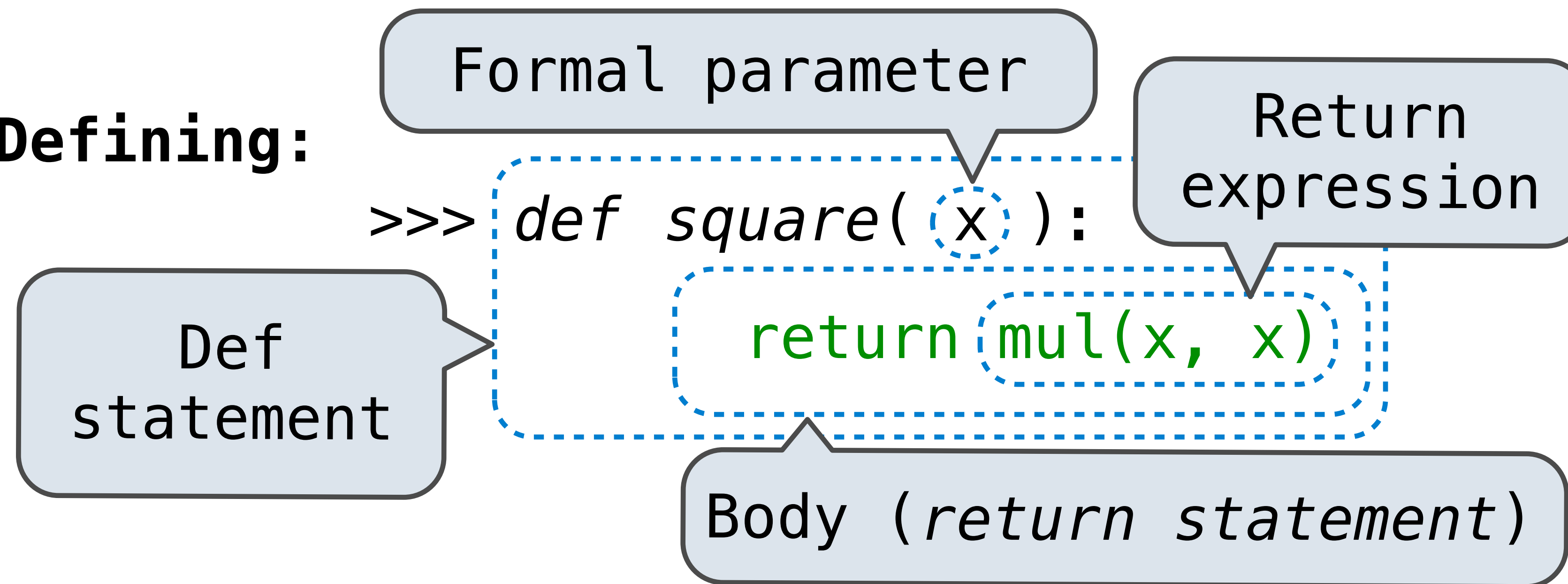
Call expression: square(2+2)

Calling/Applying:



Life Cycle of a User-Defined Function

Defining:



What happens?

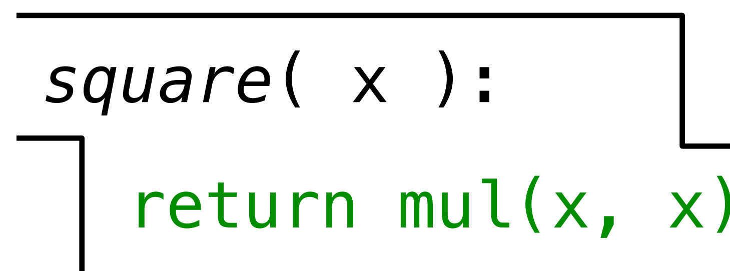
Function created

Body stored

Name bound

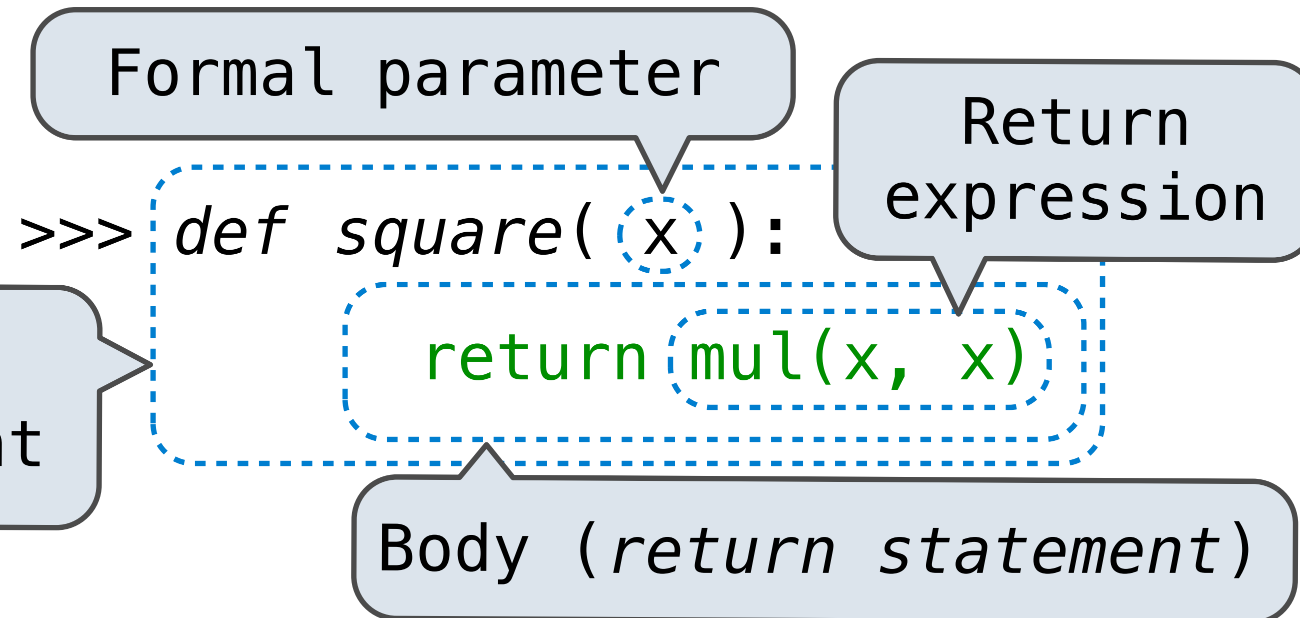
Call expression: square(2+2)

Calling/Applying:

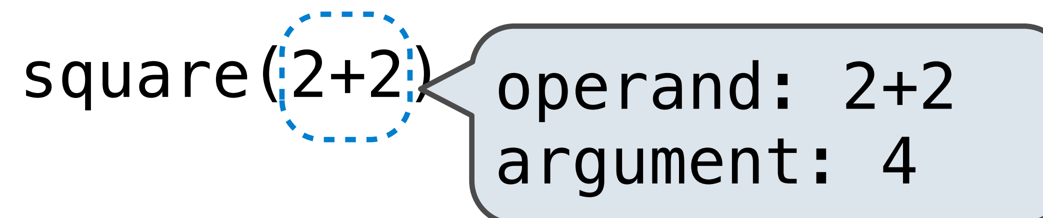


Life Cycle of a User-Defined Function

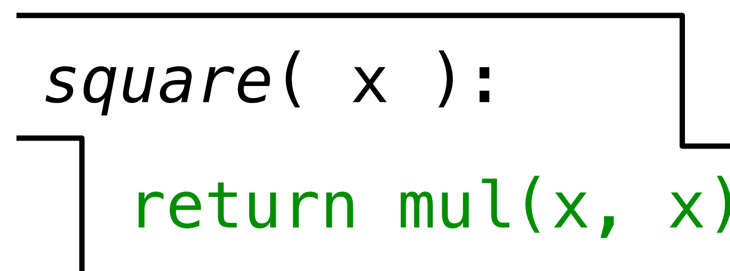
Defining:



Call expression:



Calling/Applying:



What happens?

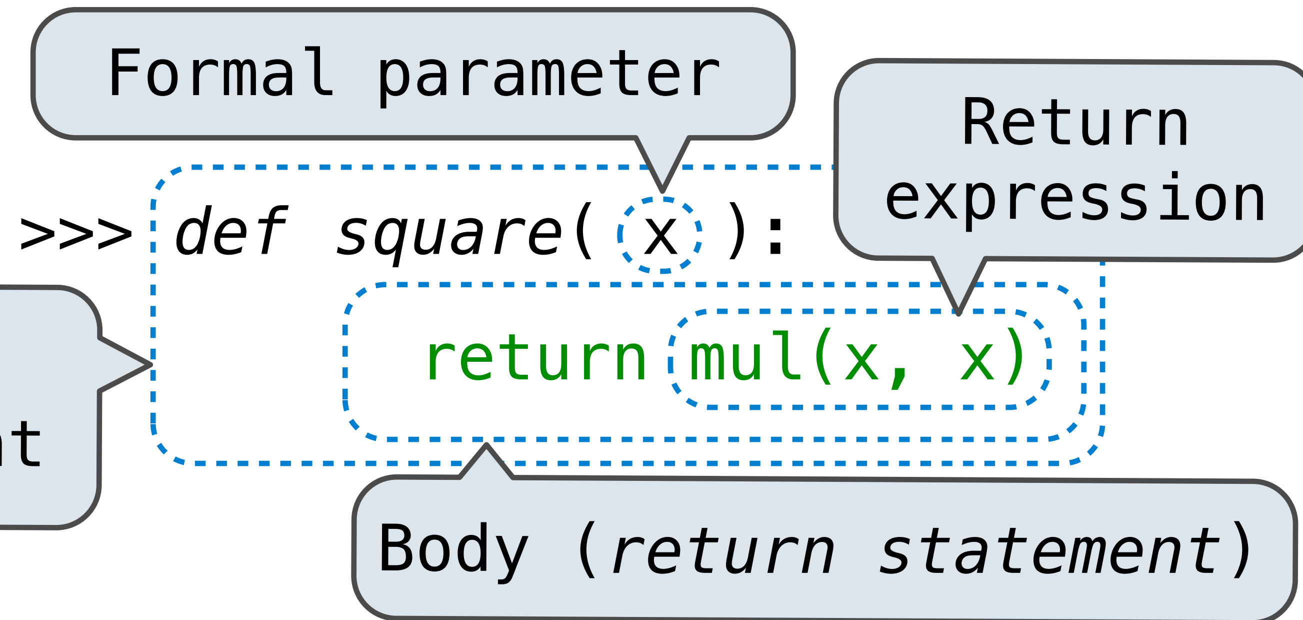
Function created

Body stored

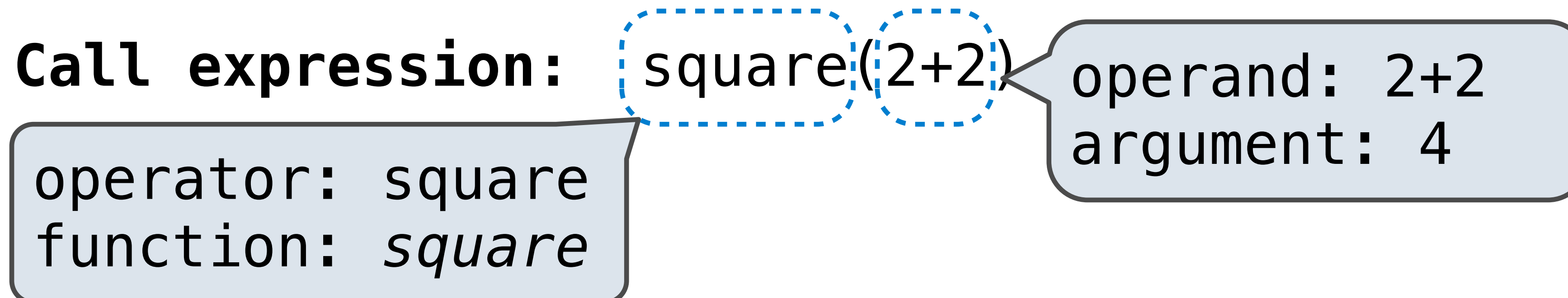
Name bound

Life Cycle of a User-Defined Function

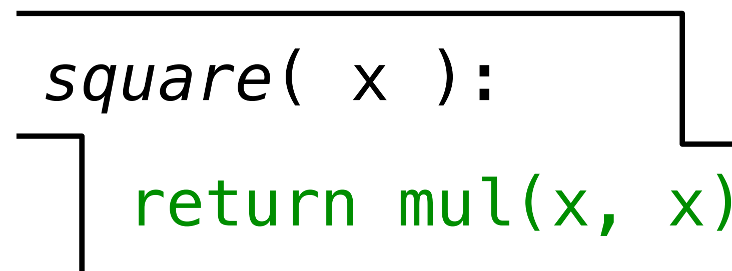
Defining:



Call expression:



Calling/Applying:



What happens?

Function created

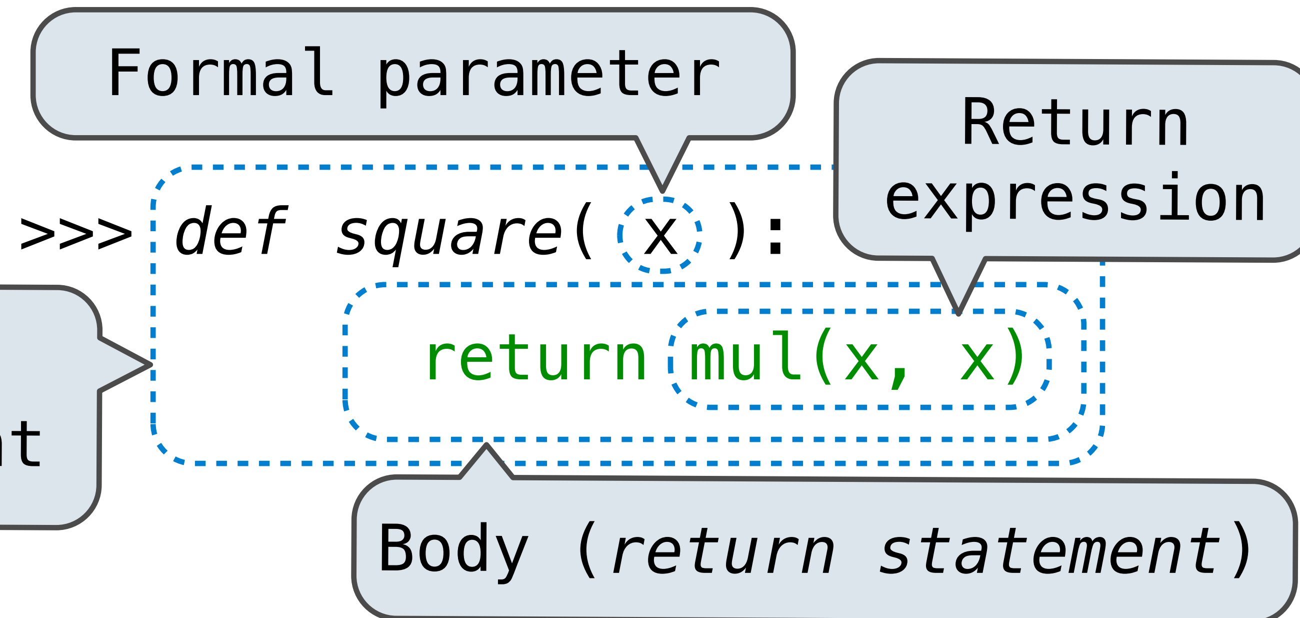
Body stored

Name bound

Life Cycle of a User-Defined Function

What happens?

Defining:

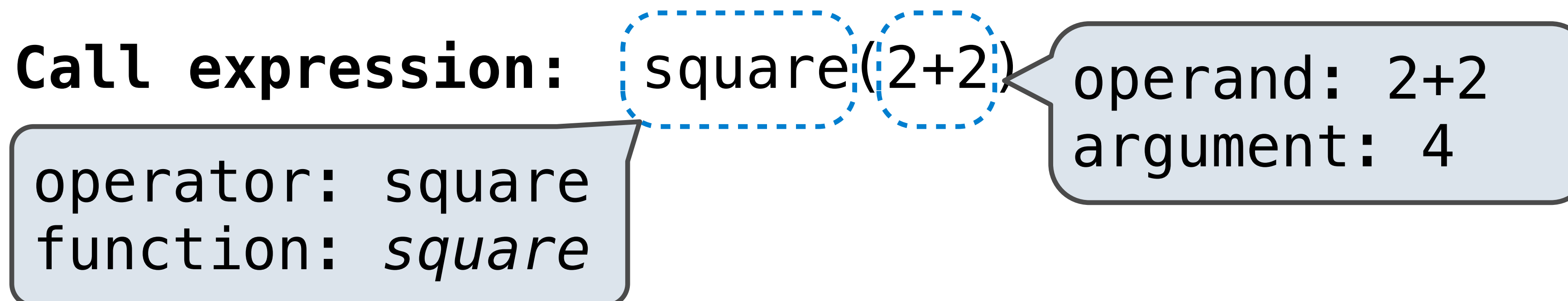


Function created

Body stored

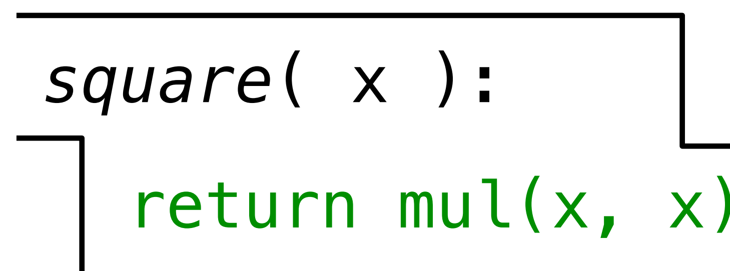
Name bound

Call expression:



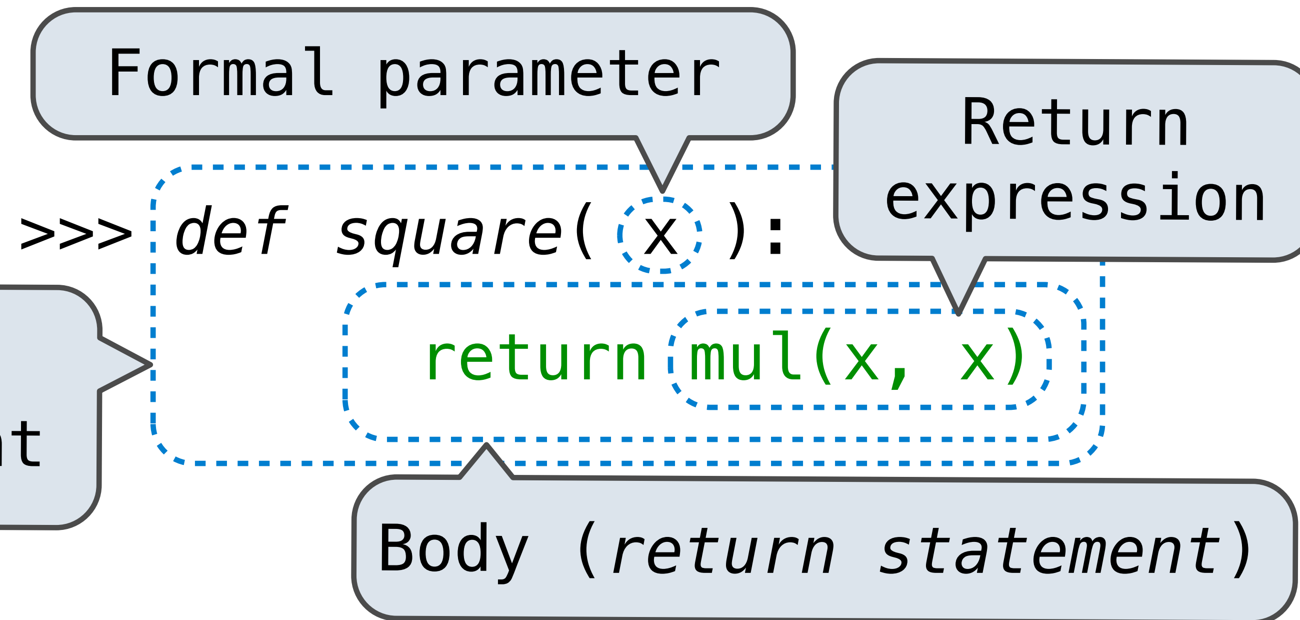
Op's evaluated

Calling/Applying:



Life Cycle of a User-Defined Function

Defining:



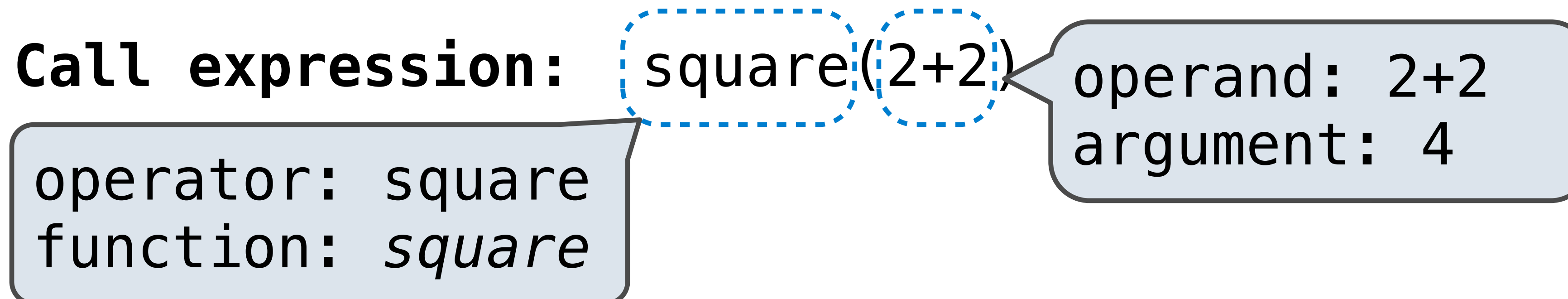
What happens?

Function created

Body stored

Name bound

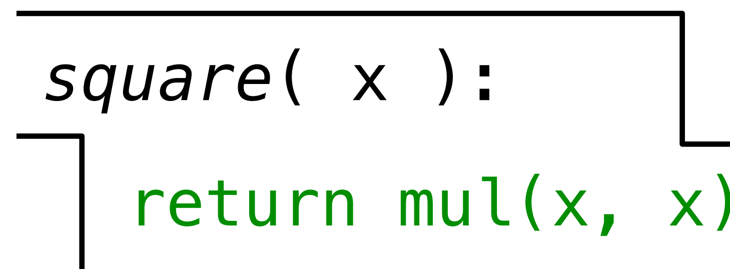
Call expression:



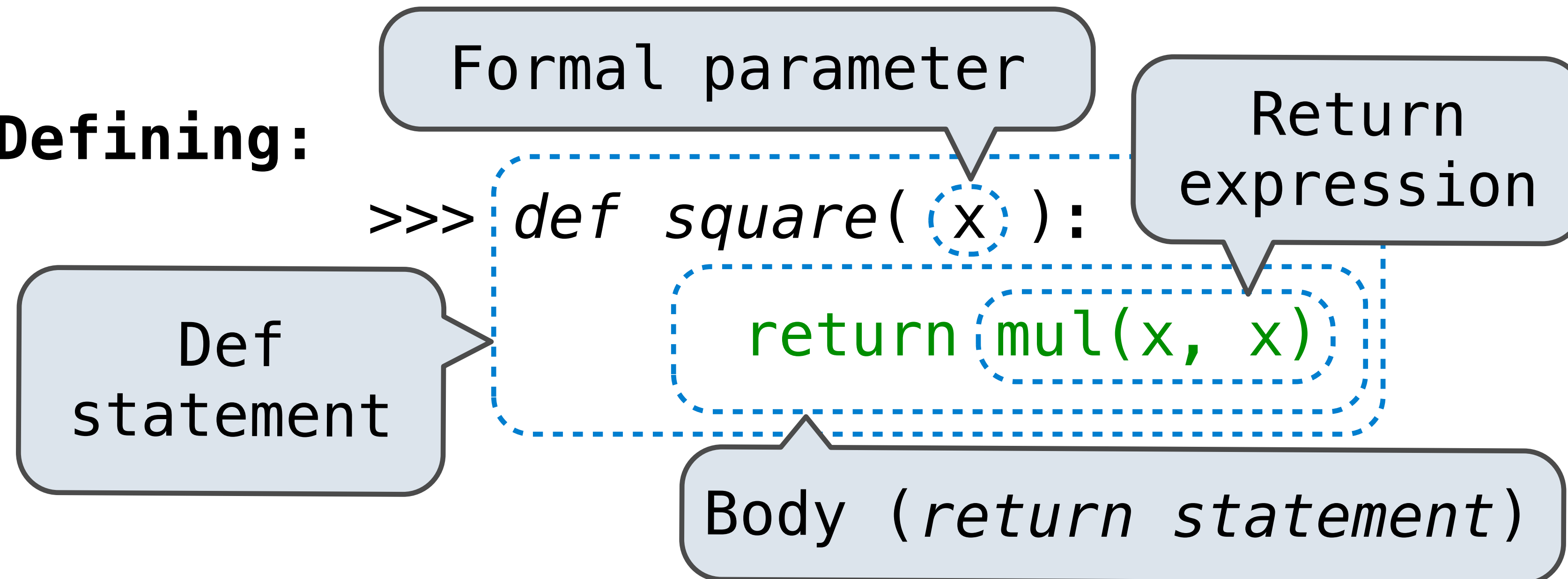
Op's evaluated

Function called

Calling/Applying:



Life Cycle of a User-Defined Function

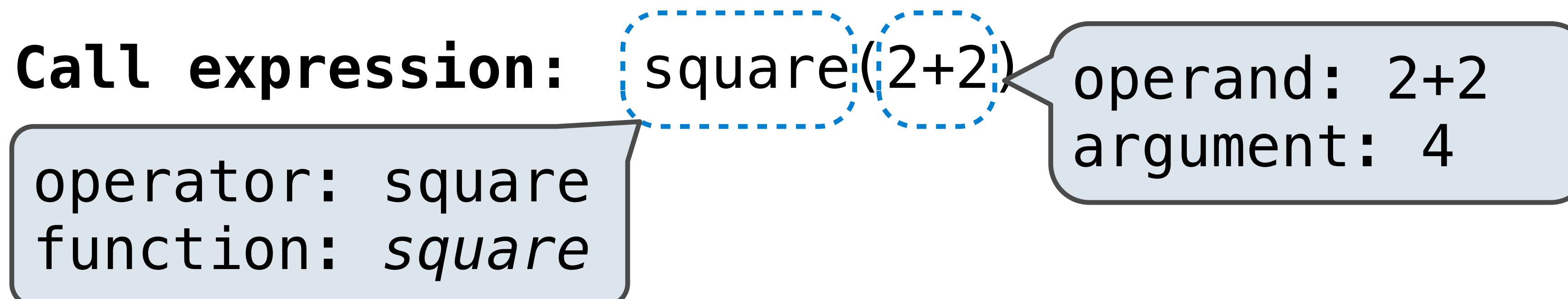


What happens?

Function created

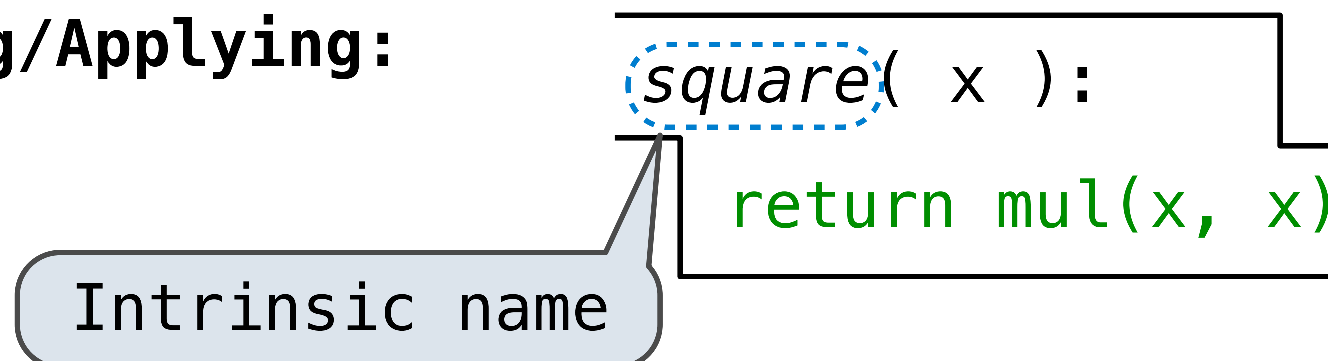
Body stored

Name bound

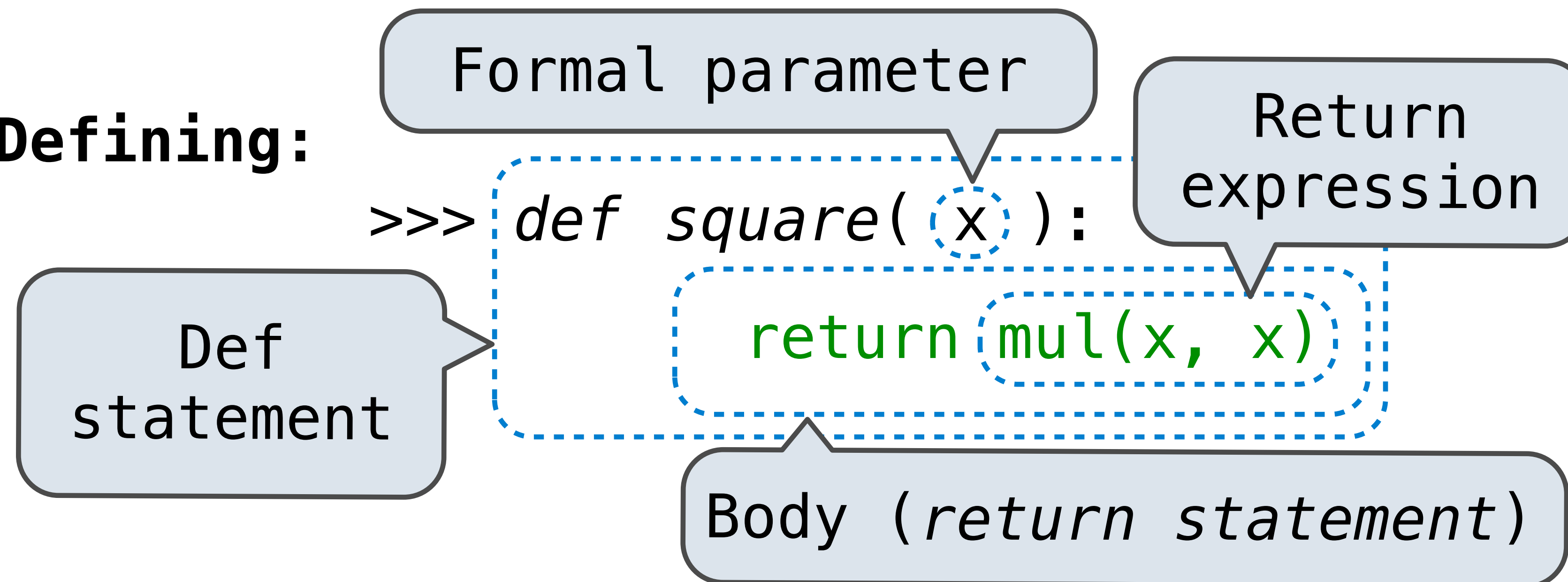


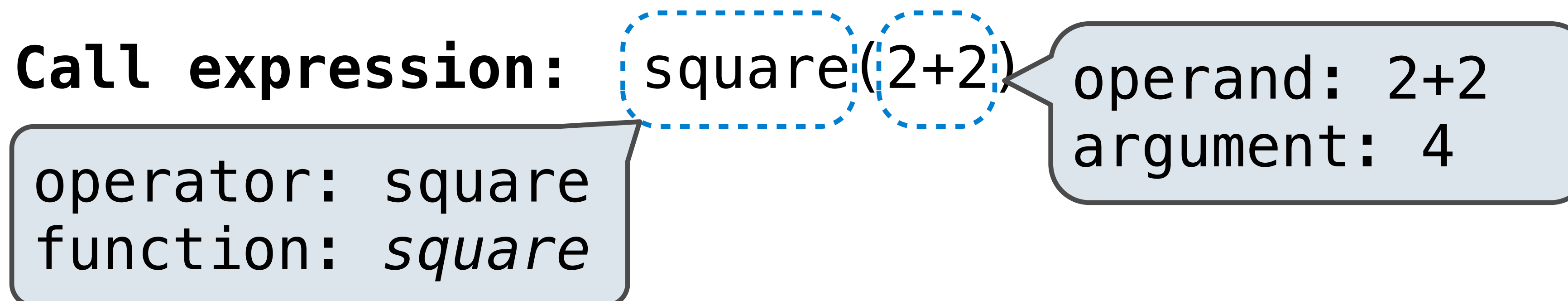
Op's evaluated

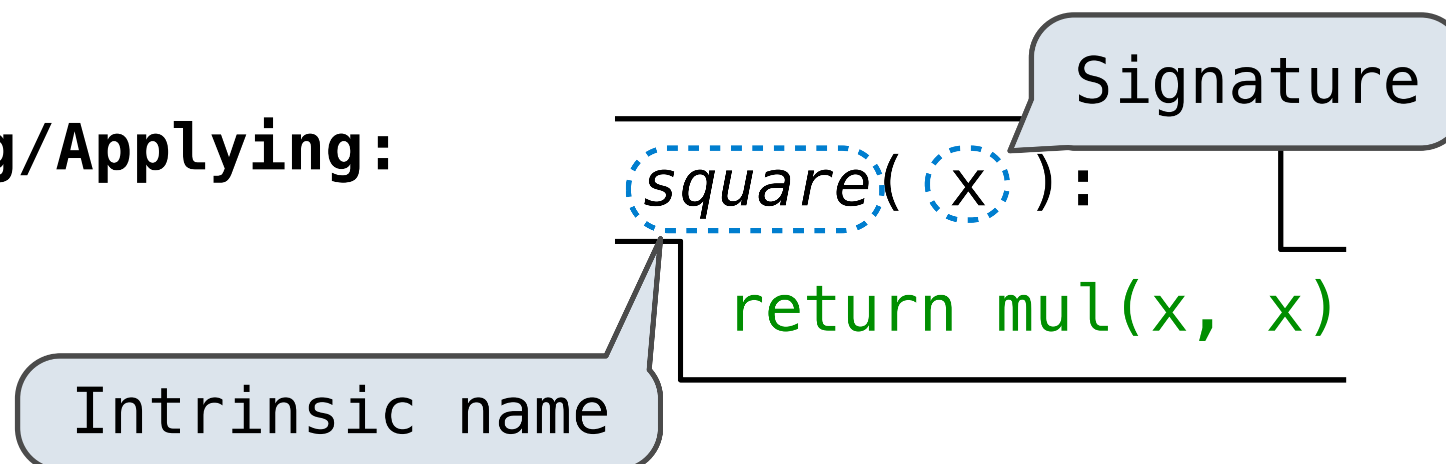
Function called



Life Cycle of a User-Defined Function







What happens?

Function created

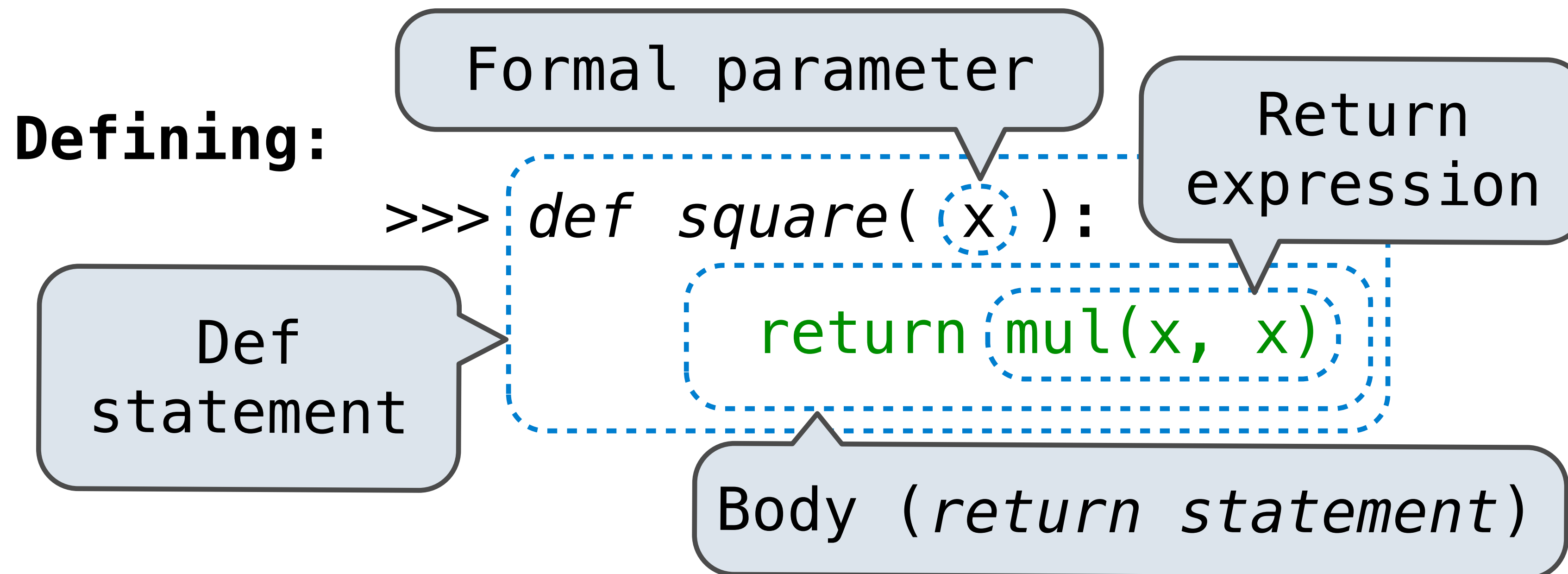
Body stored

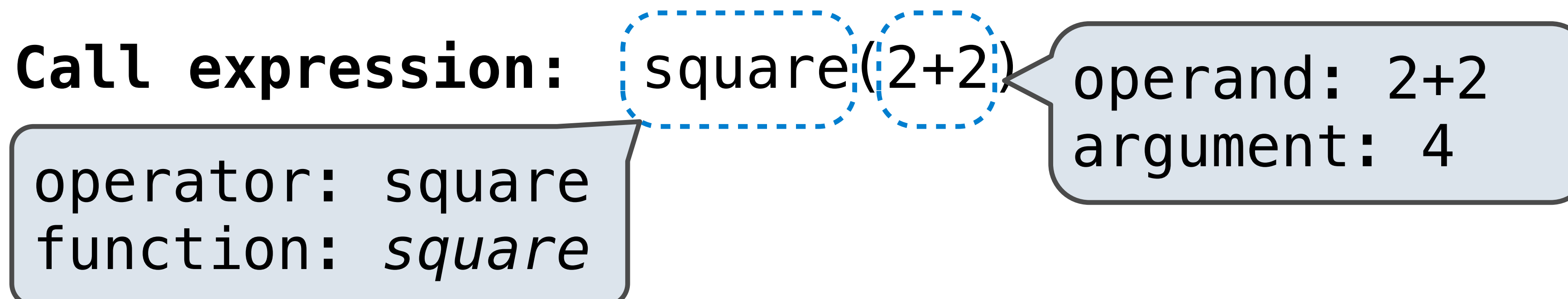
Name bound

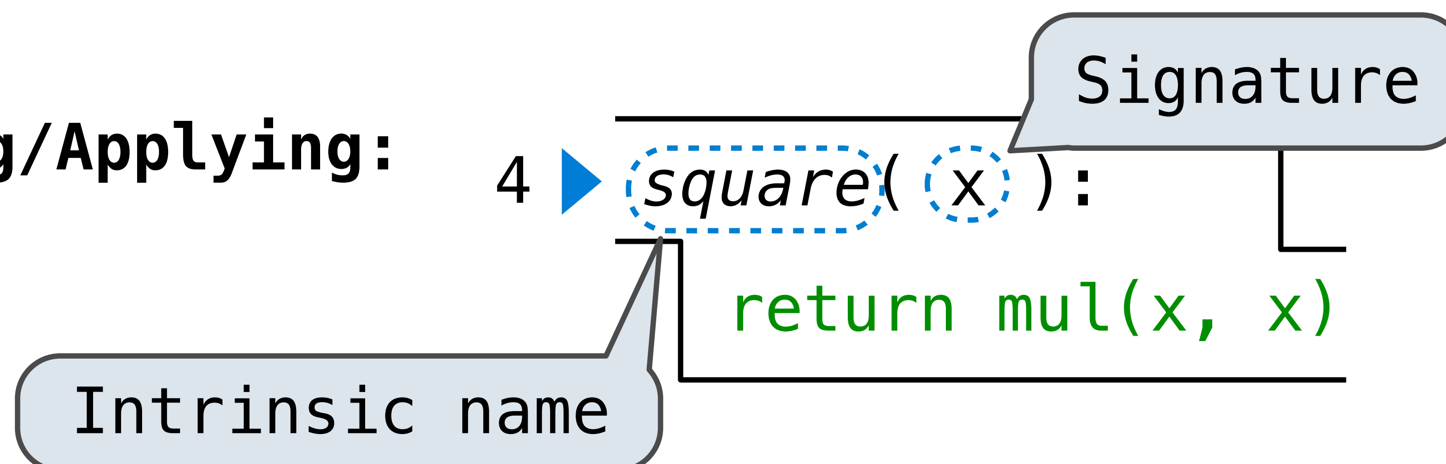
Op's evaluated

Function called

Life Cycle of a User-Defined Function







What happens?

Function created

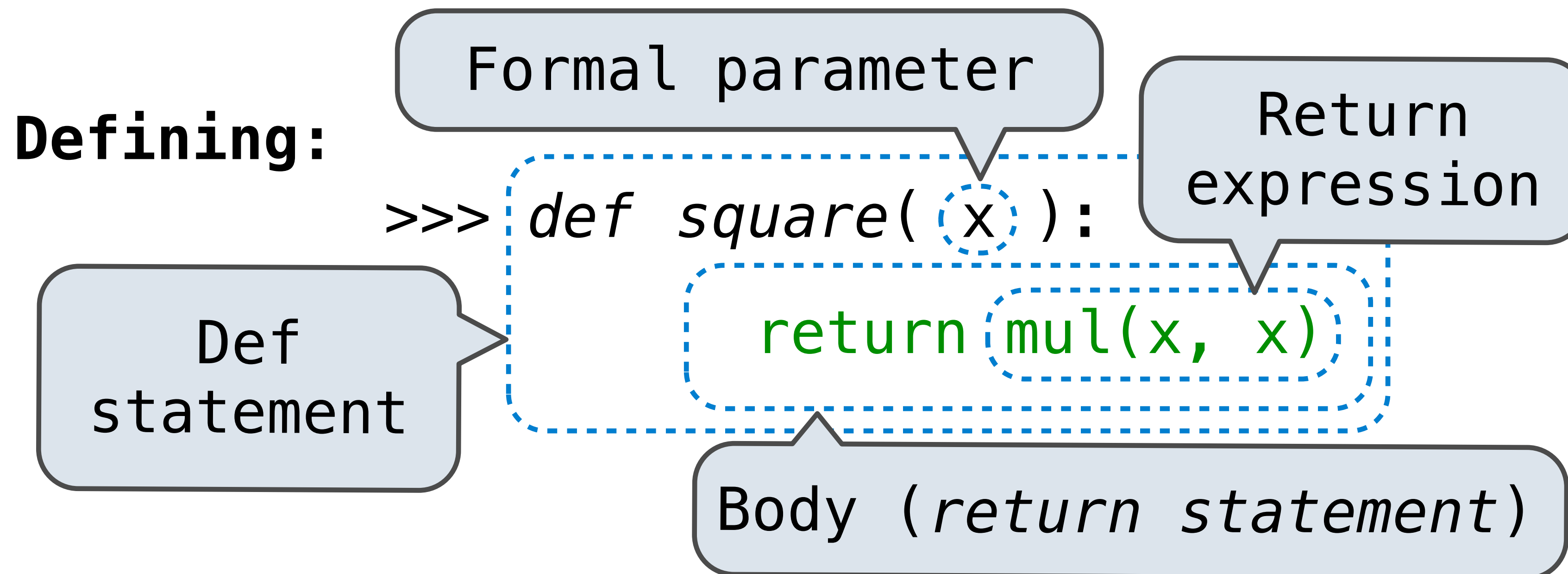
Body stored

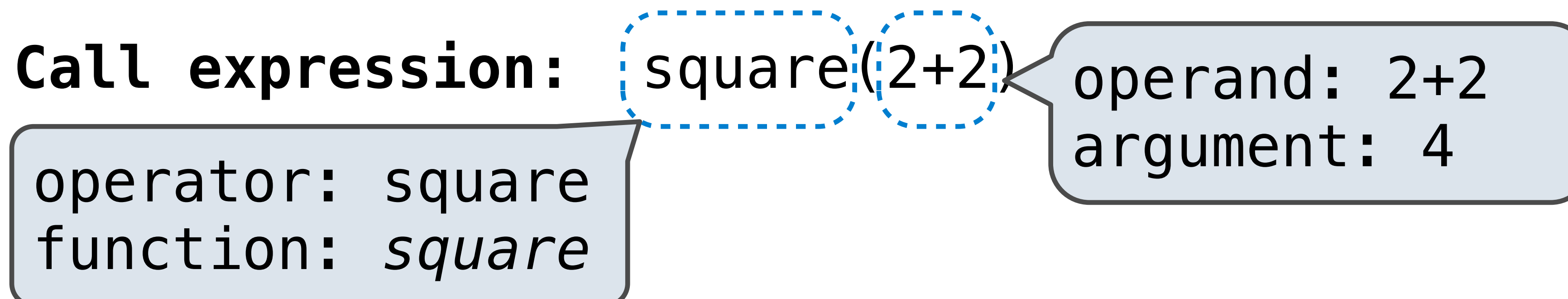
Name bound

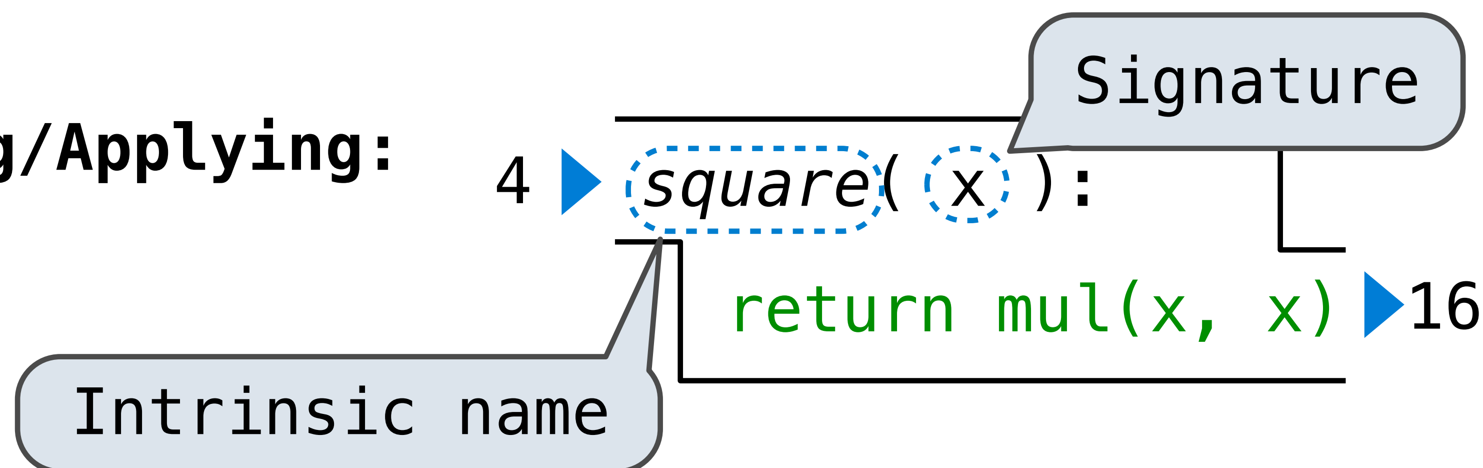
Op's evaluated

Function called

Life Cycle of a User-Defined Function







What happens?

Function created

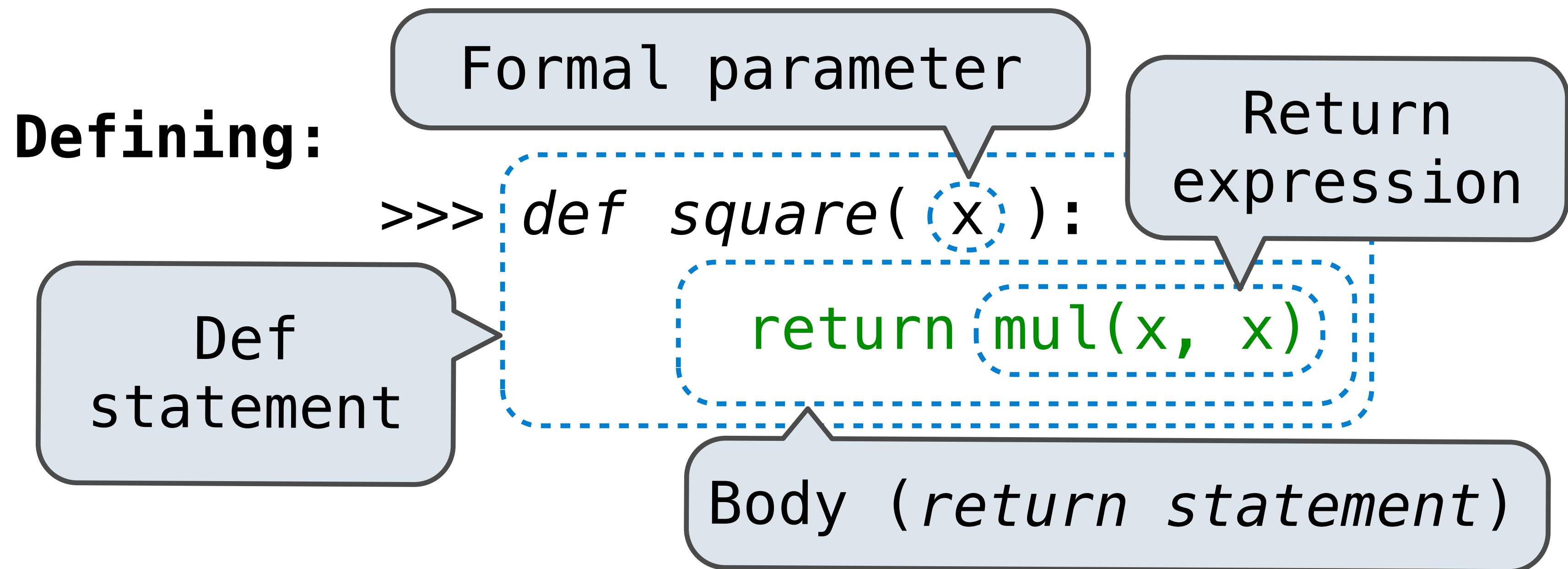
Body stored

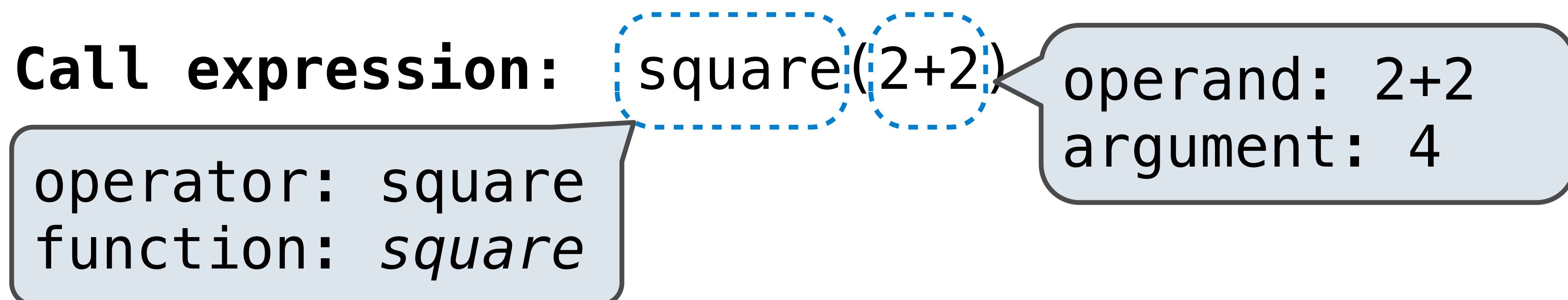
Name bound

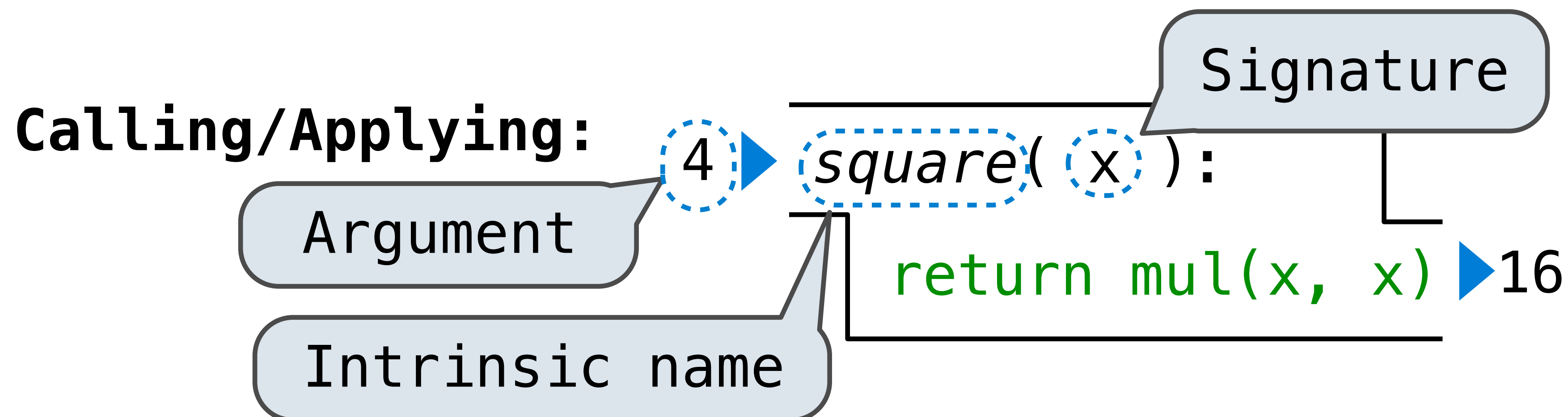
Op's evaluated

Function called

Life Cycle of a User-Defined Function







What happens?

Function created

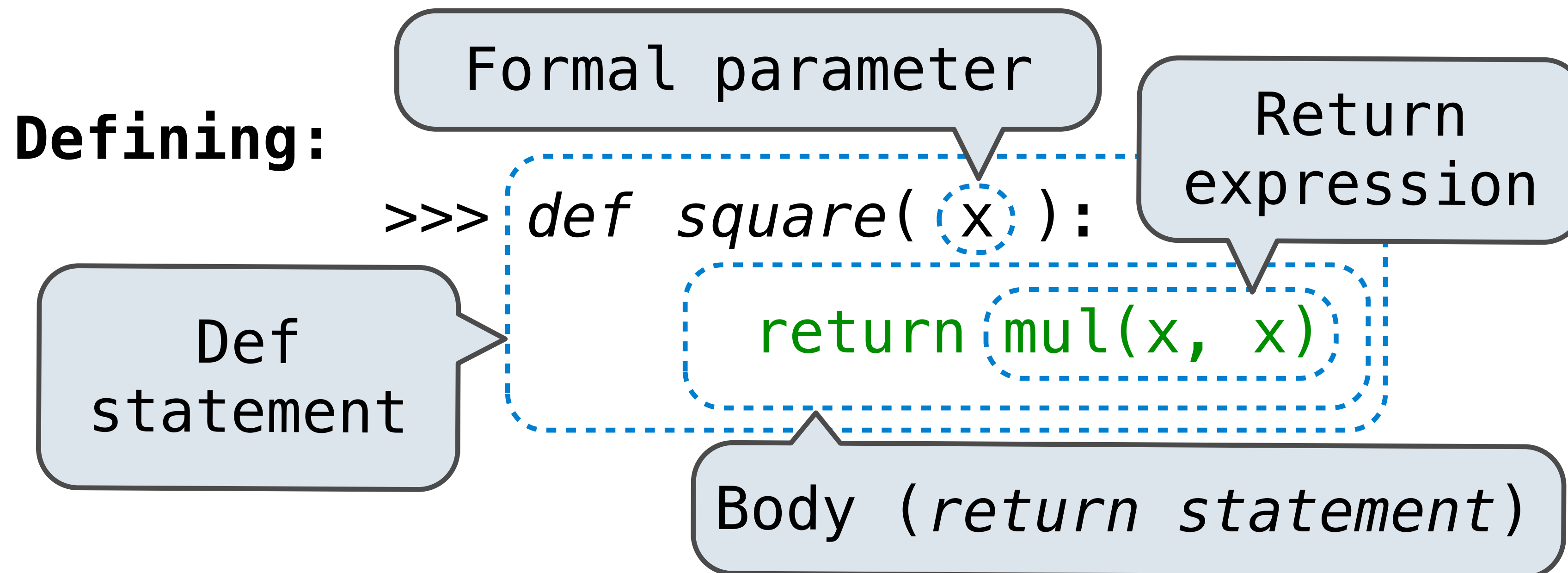
Body stored

Name bound

Op's evaluated

Function called

Life Cycle of a User-Defined Function

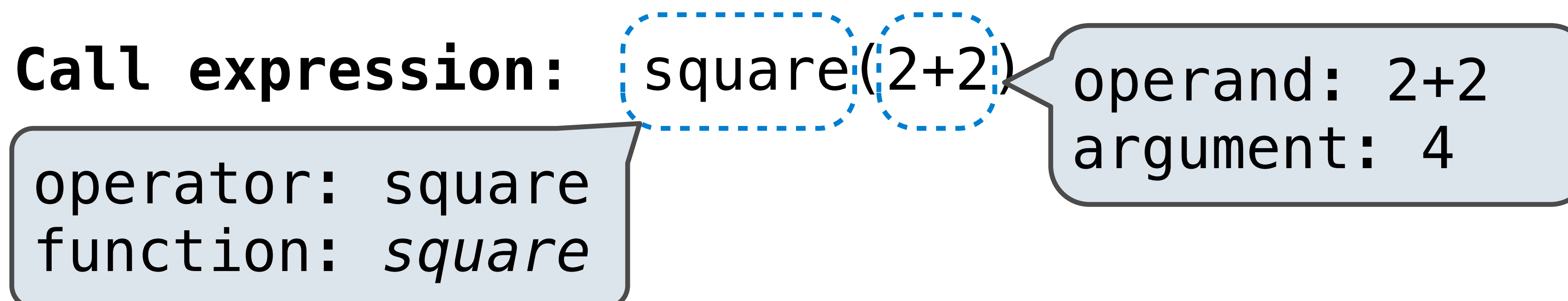


What happens?

Function created

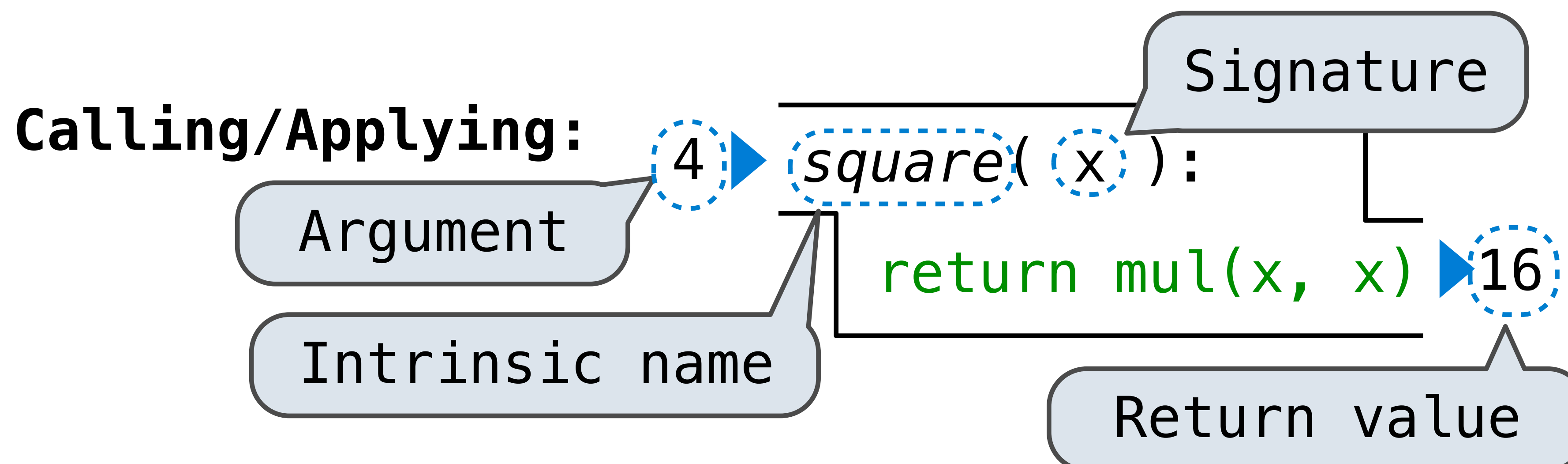
Body stored

Name bound

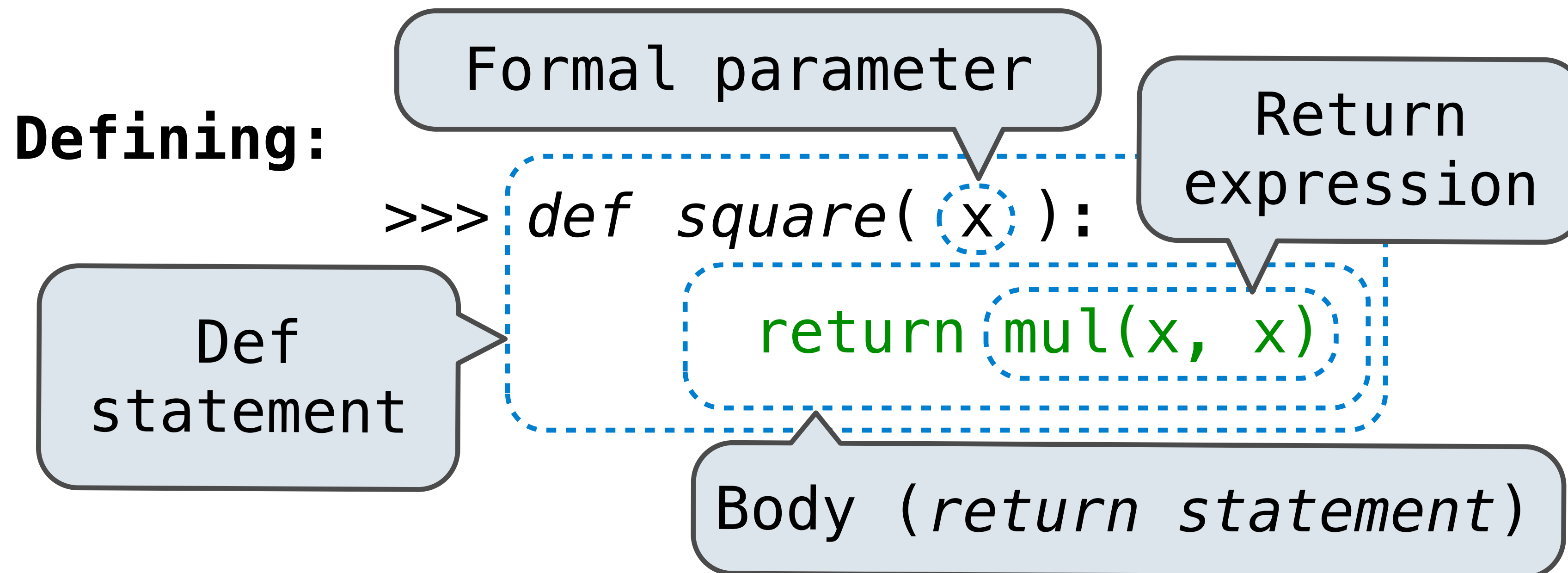


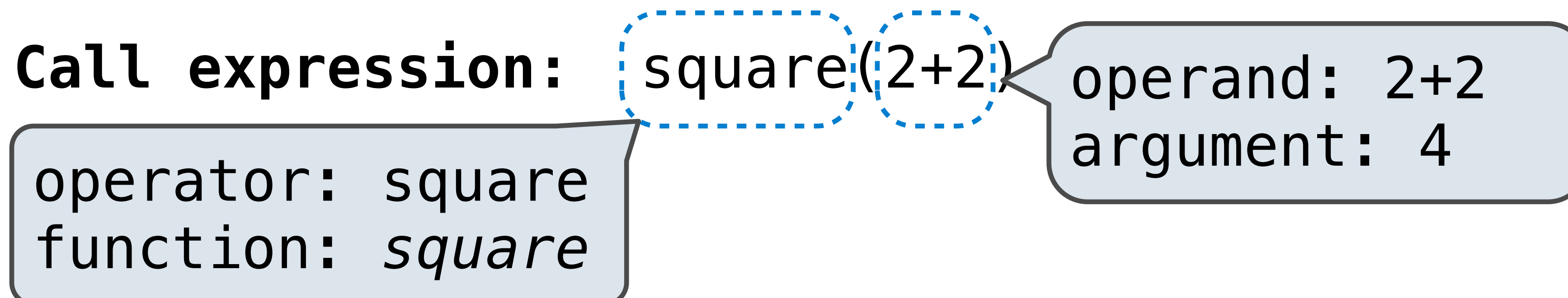
Op's evaluated

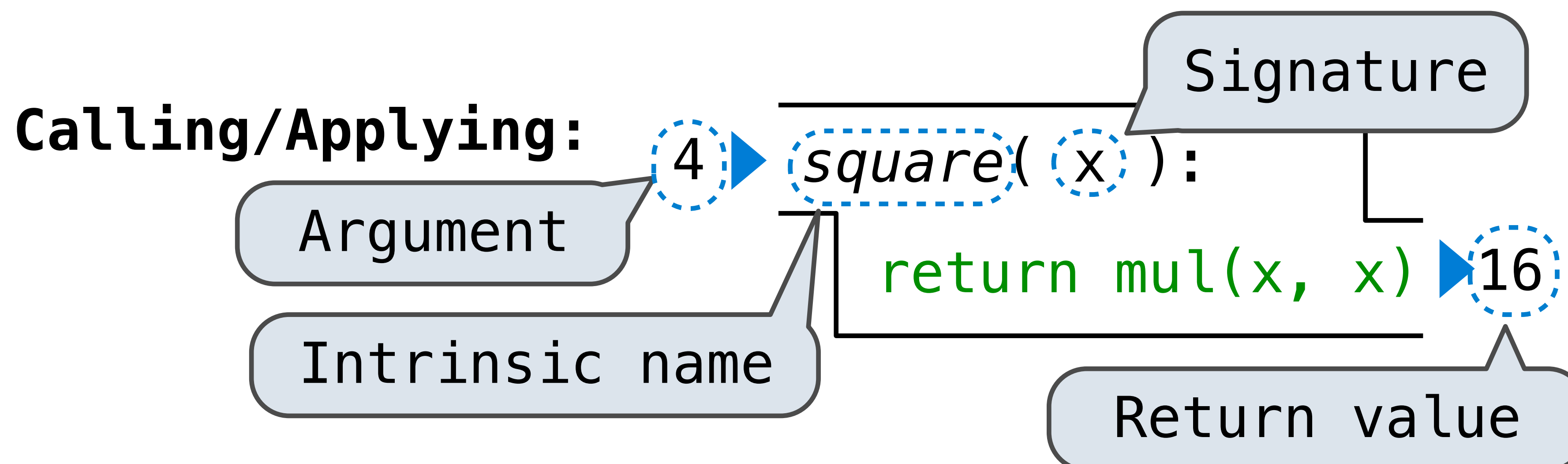
Function called



Life Cycle of a User-Defined Function







What happens?

Function created

Body stored

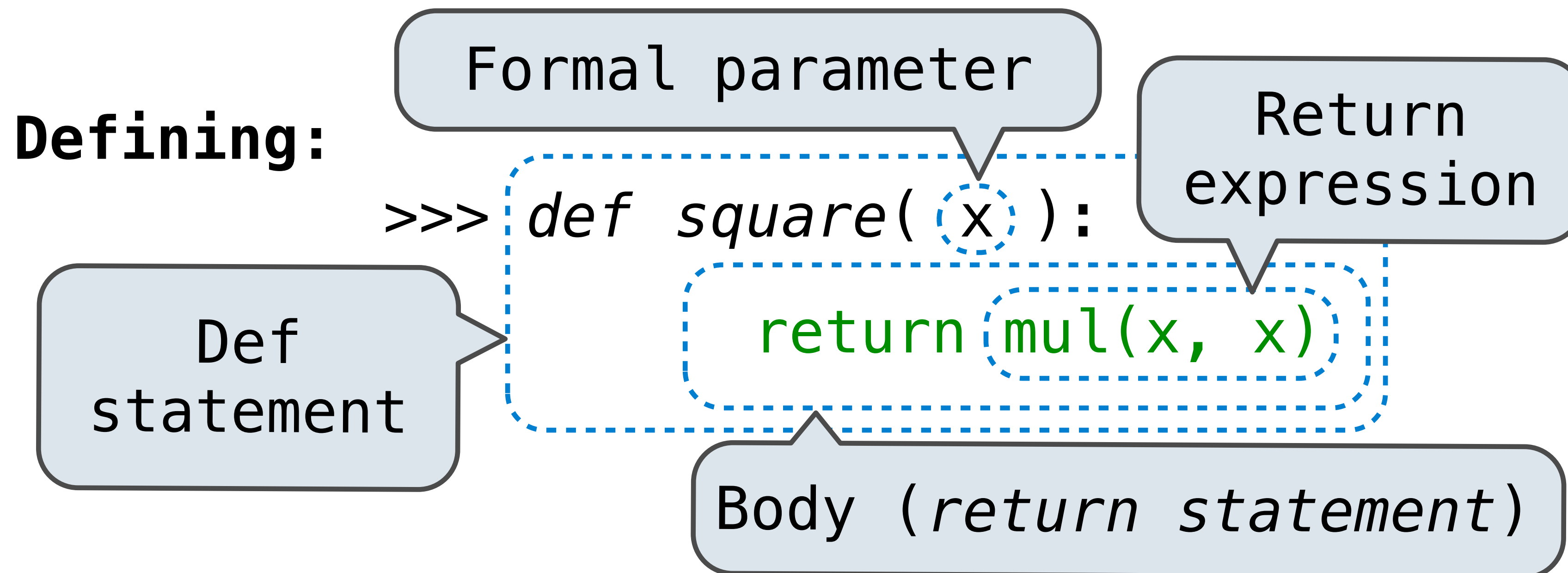
Name bound

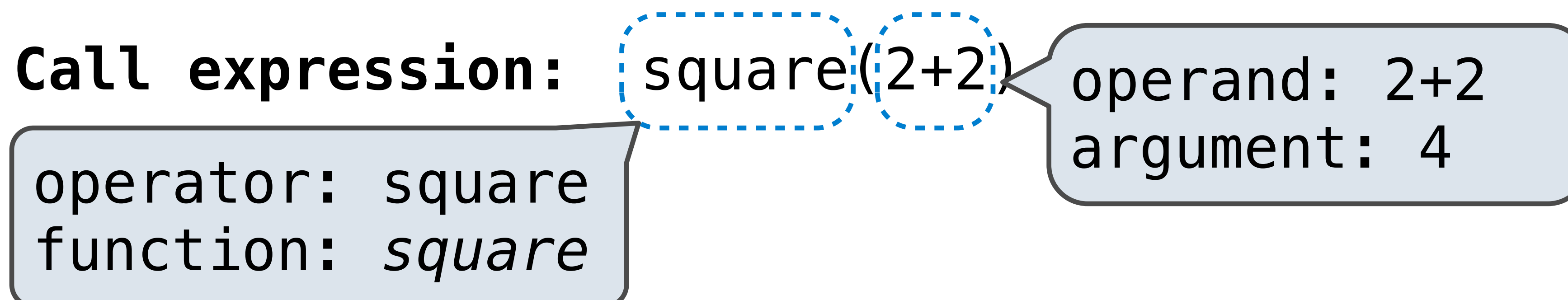
Op's evaluated

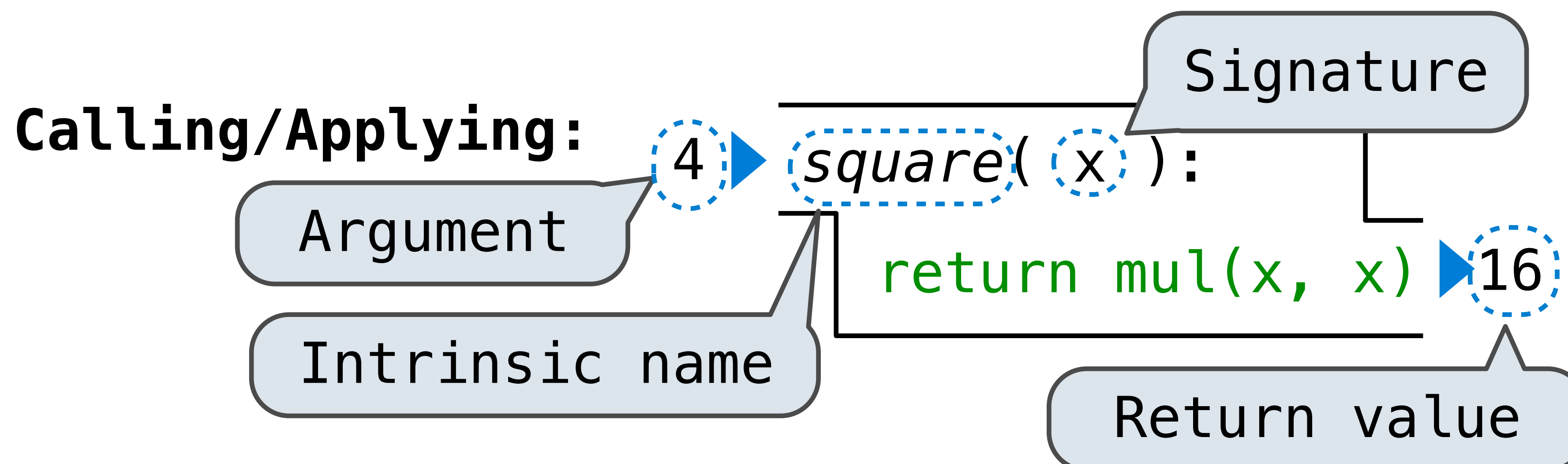
Function called

New frame!

Life Cycle of a User-Defined Function







What happens?

Function created

Body stored

Name bound

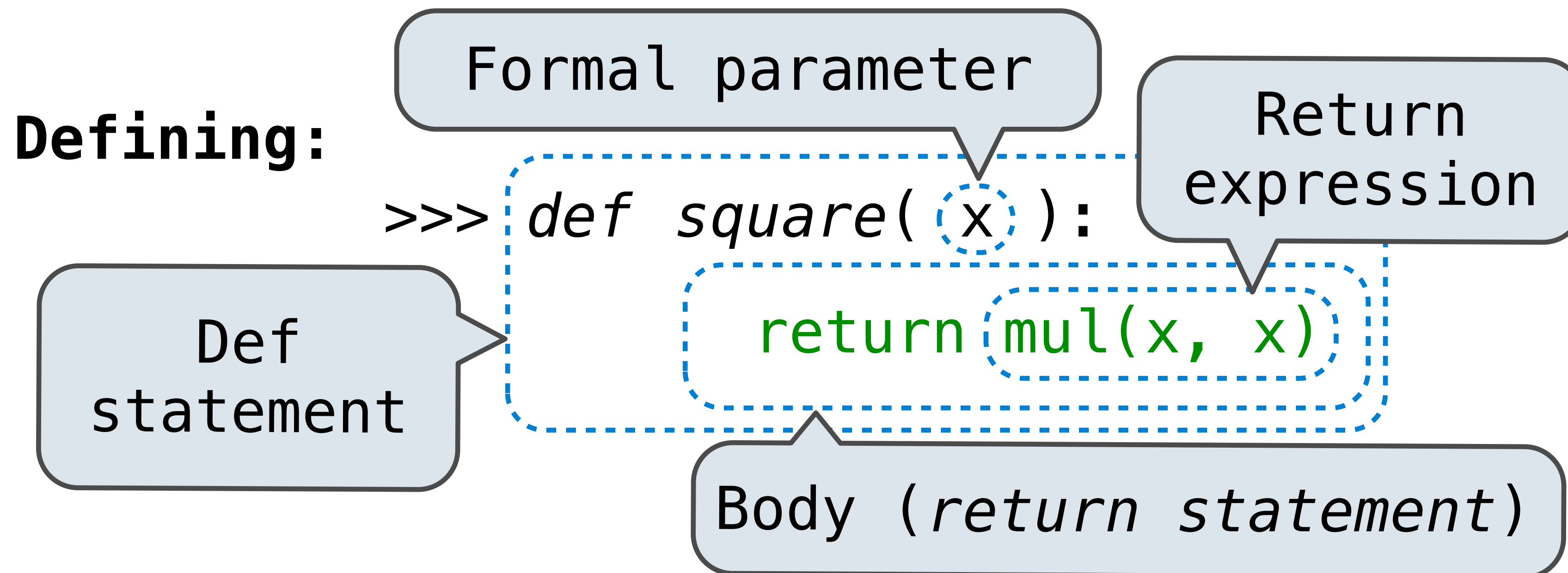
Op's evaluated

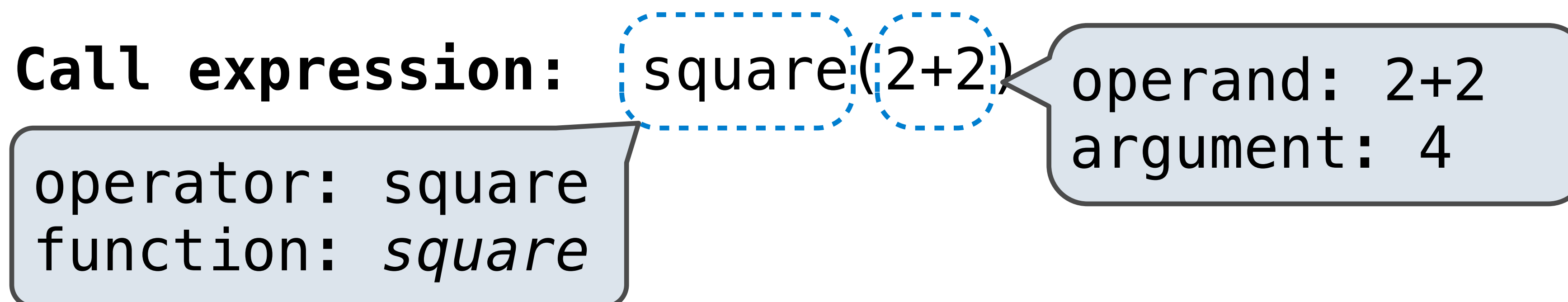
Function called

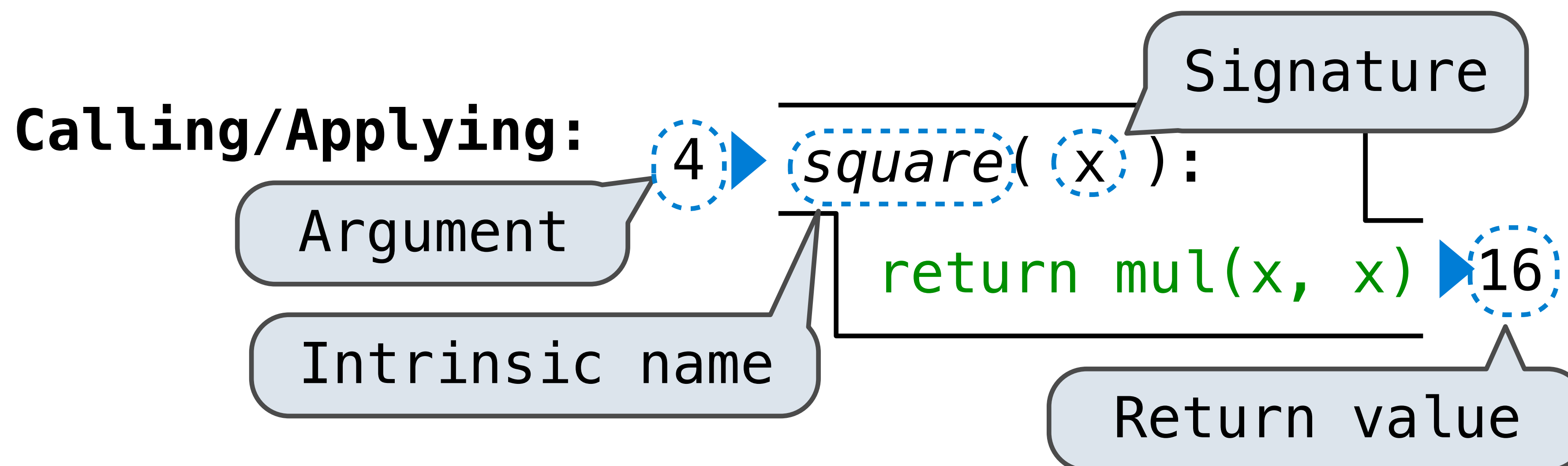
New frame!

Params bound

Life Cycle of a User-Defined Function







What happens?

Function created

Body stored

Name bound

Op's evaluated

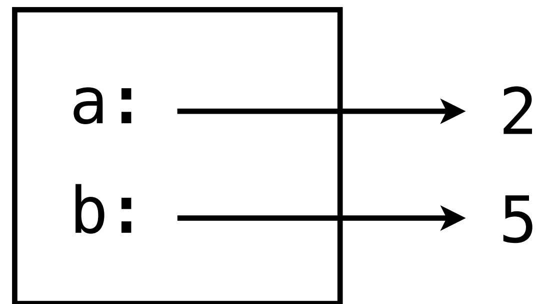
Function called

New frame!

Params bound

Body evaluated

Cast of Characters: Environment Diagrams

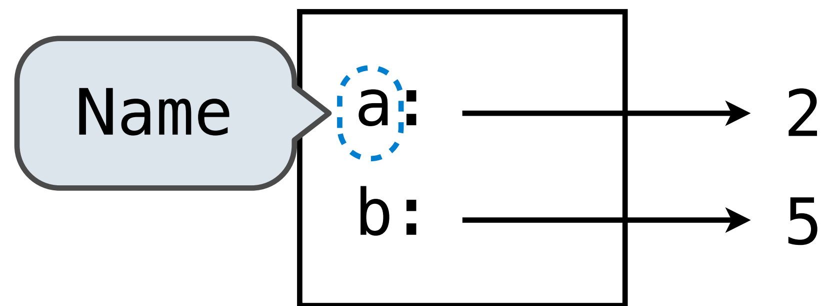


Frames:

Environments:

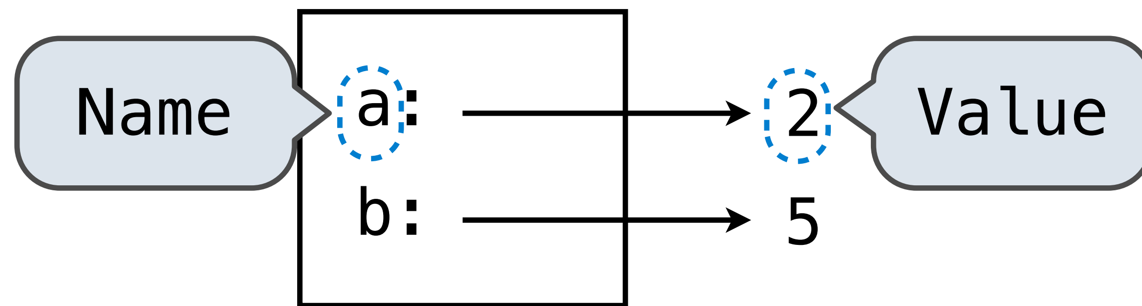
Cast of Characters: Environment Diagrams

Frames:



Environments:

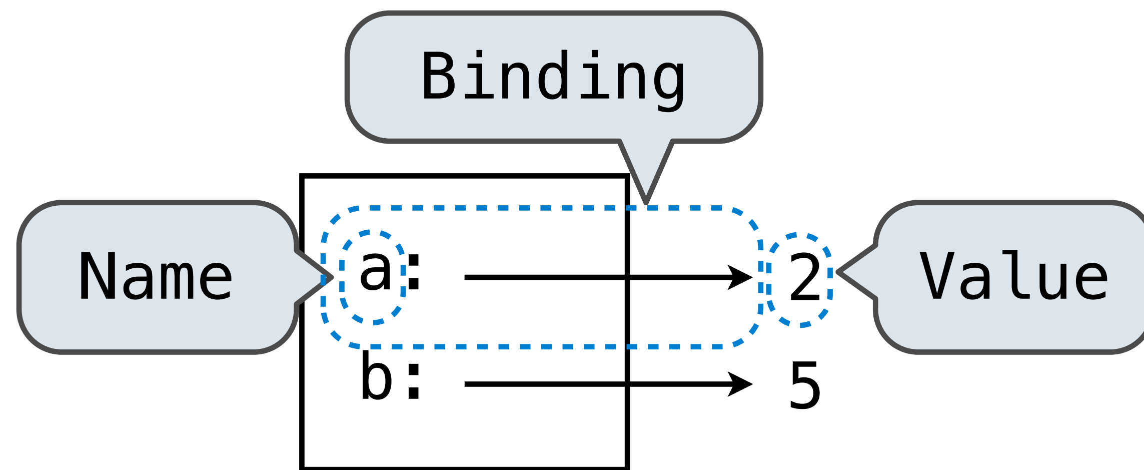
Cast of Characters: Environment Diagrams



Frames:

Environments:

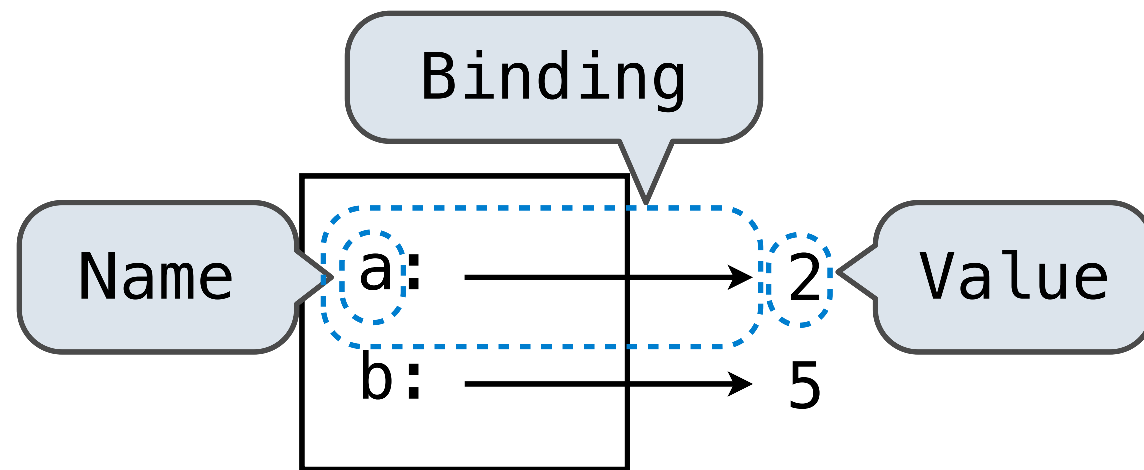
Cast of Characters: Environment Diagrams



Frames:

Environments:

Cast of Characters: Environment Diagrams

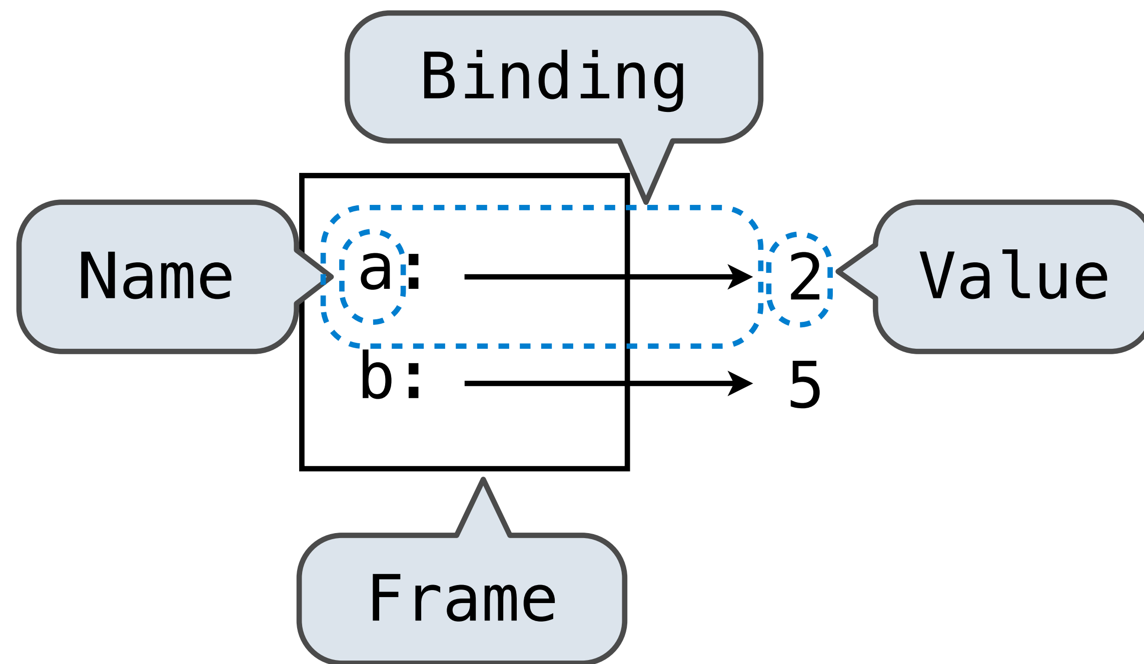


Frames:

A name is bound to a value

Environments:

Cast of Characters: Environment Diagrams

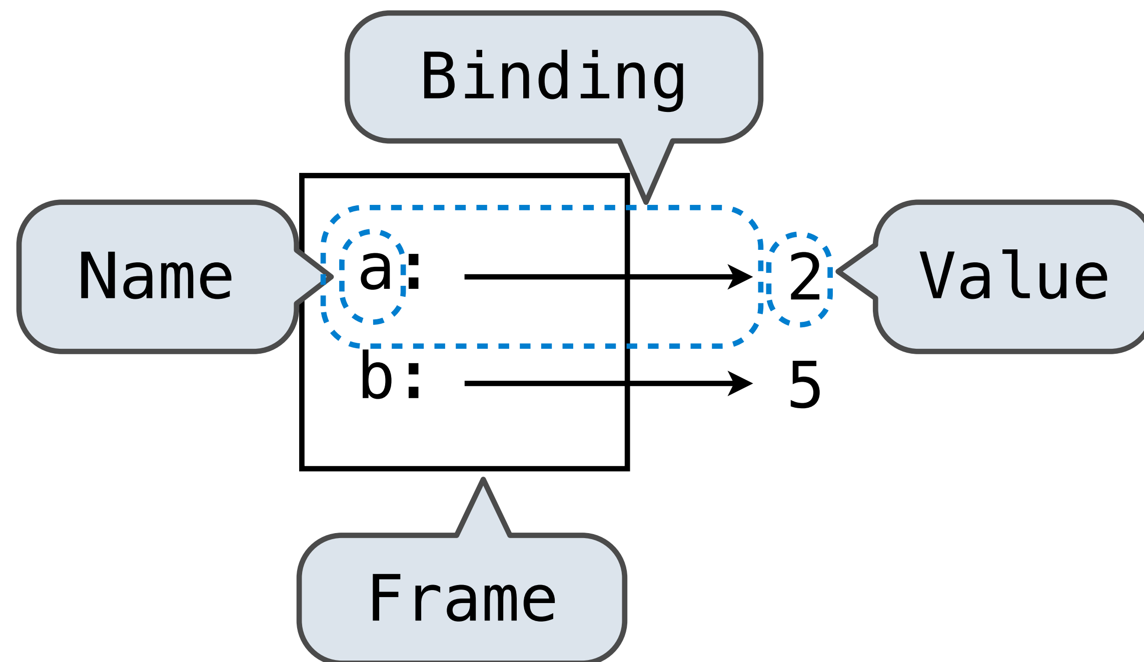


Frames:

A name is bound to a value

Environments:

Cast of Characters: Environment Diagrams



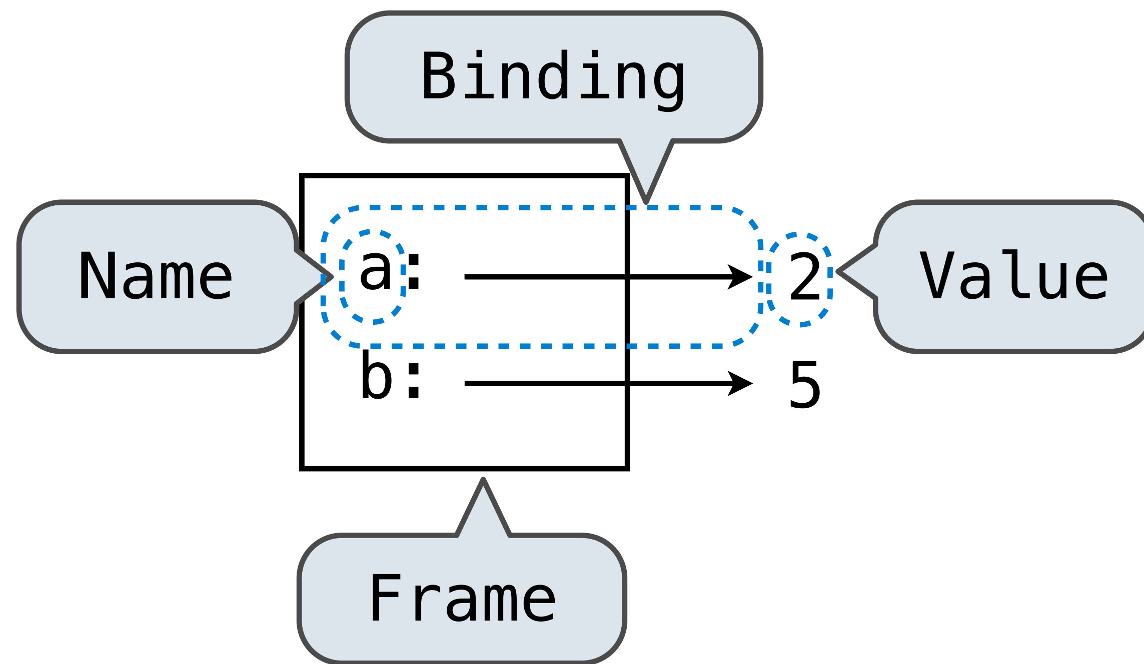
Frames:

A name is bound to a value

A frame is a rectangle
that contains bindings

Environments:

Cast of Characters: Environment Diagrams



Frames:

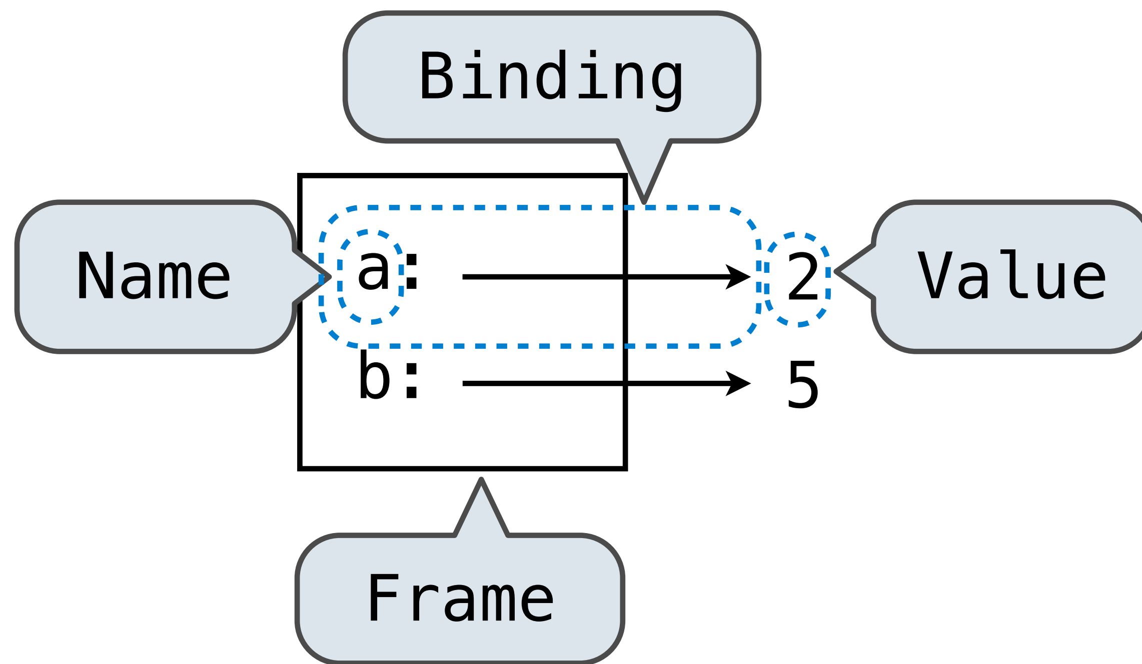
A name is bound to a value

A frame is a rectangle
that contains bindings

In a frame, there is at most
one binding per name

Environments:

Cast of Characters: Environment Diagrams

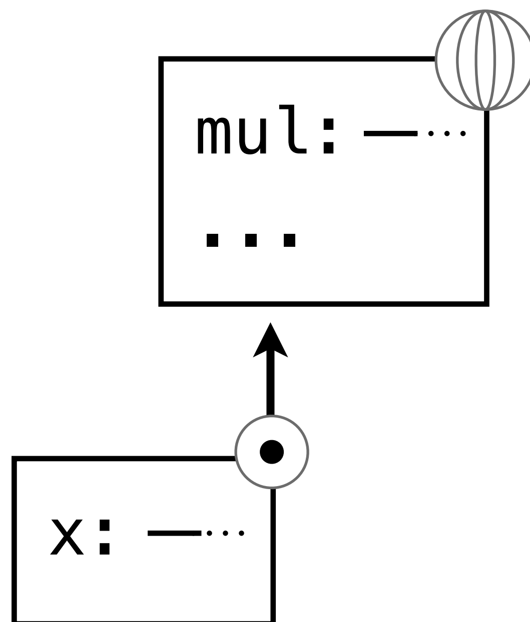


Frames:

A name is bound to a value

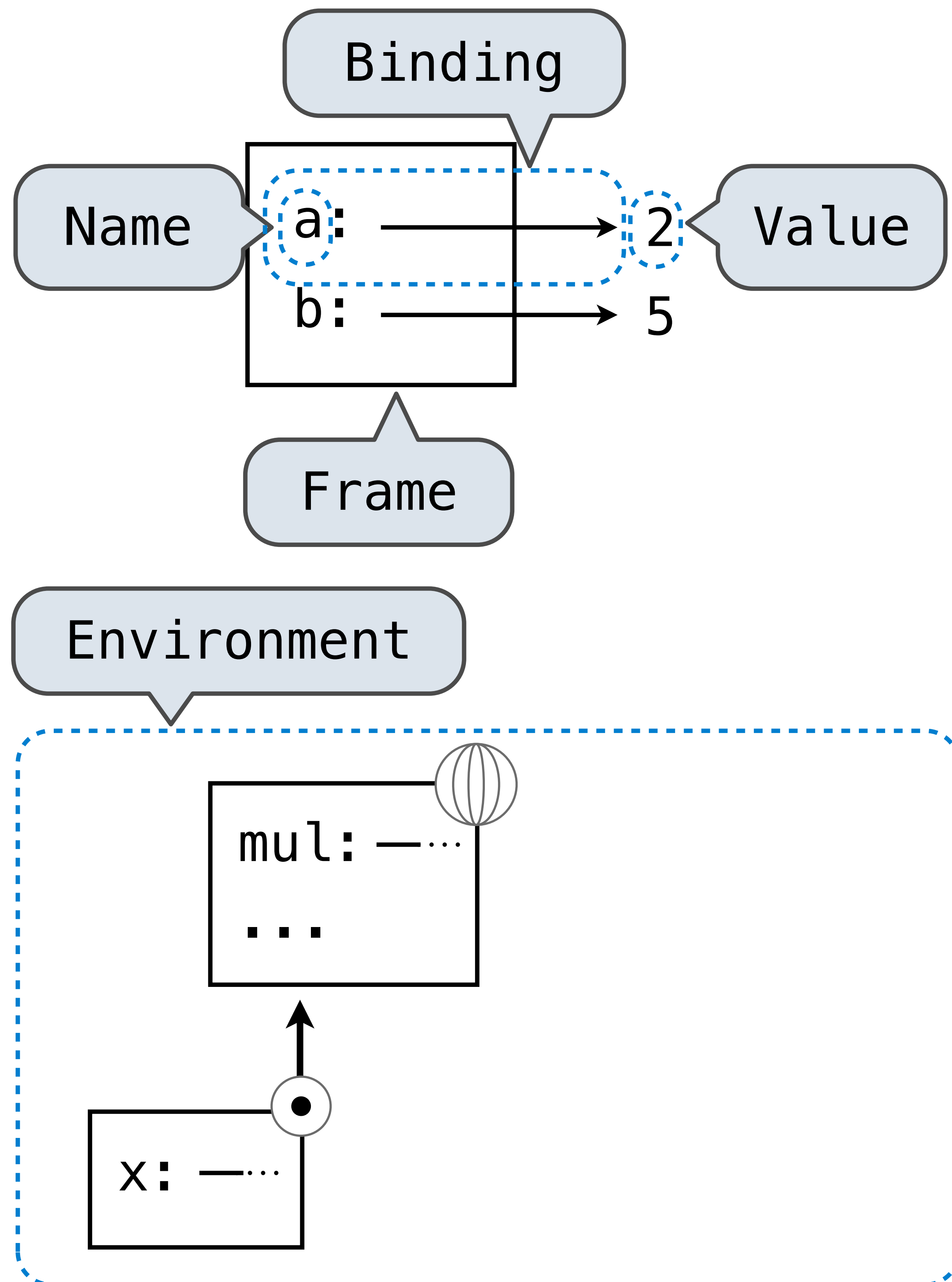
A frame is a rectangle that contains bindings

In a frame, there is at most one binding per name



Environments:

Cast of Characters: Environment Diagrams



Frames:

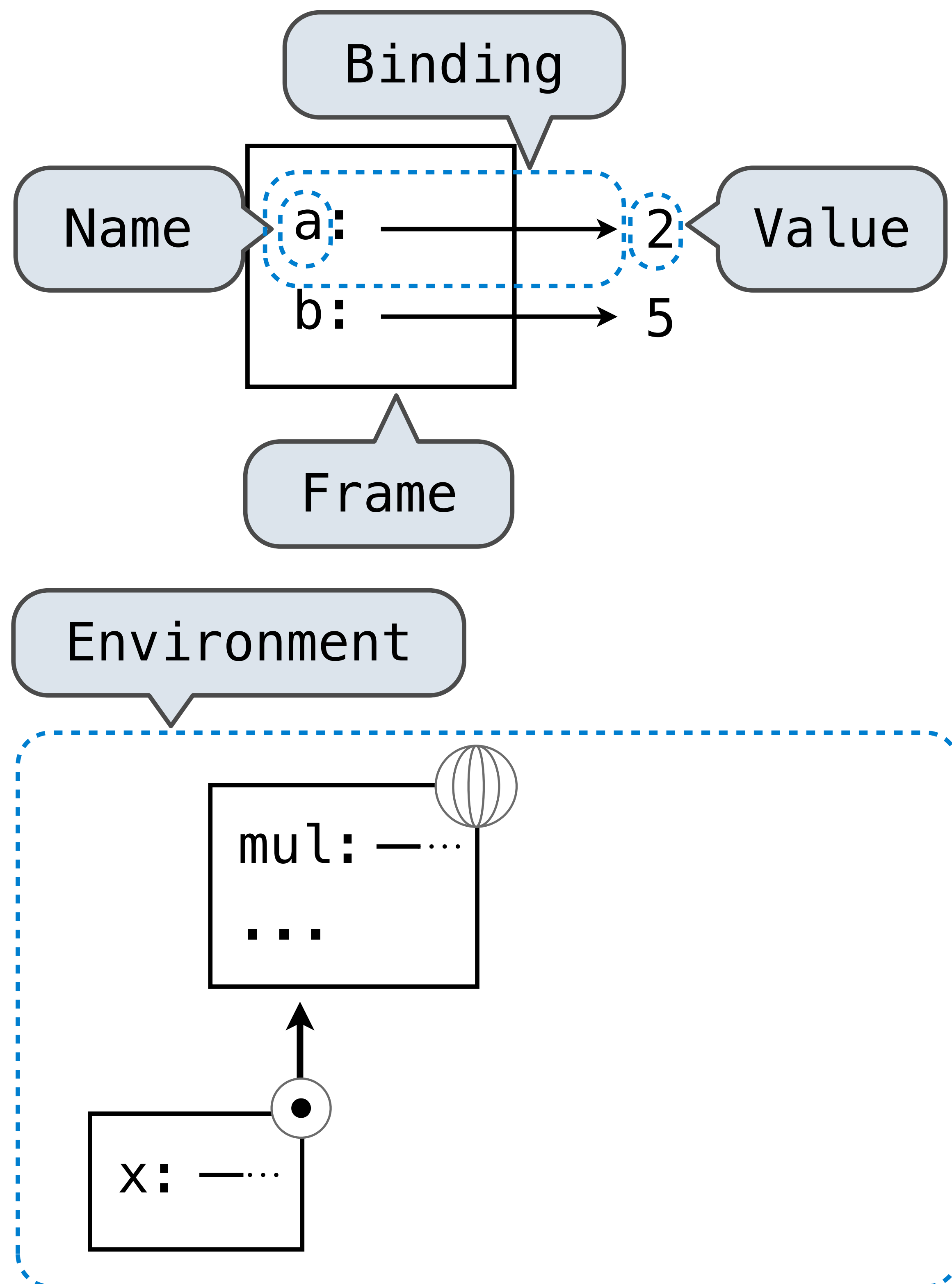
A name is bound to a value

A frame is a rectangle that contains bindings

In a frame, there is at most one binding per name

Environments:

Cast of Characters: Environment Diagrams



Frames:

A name is bound to a value

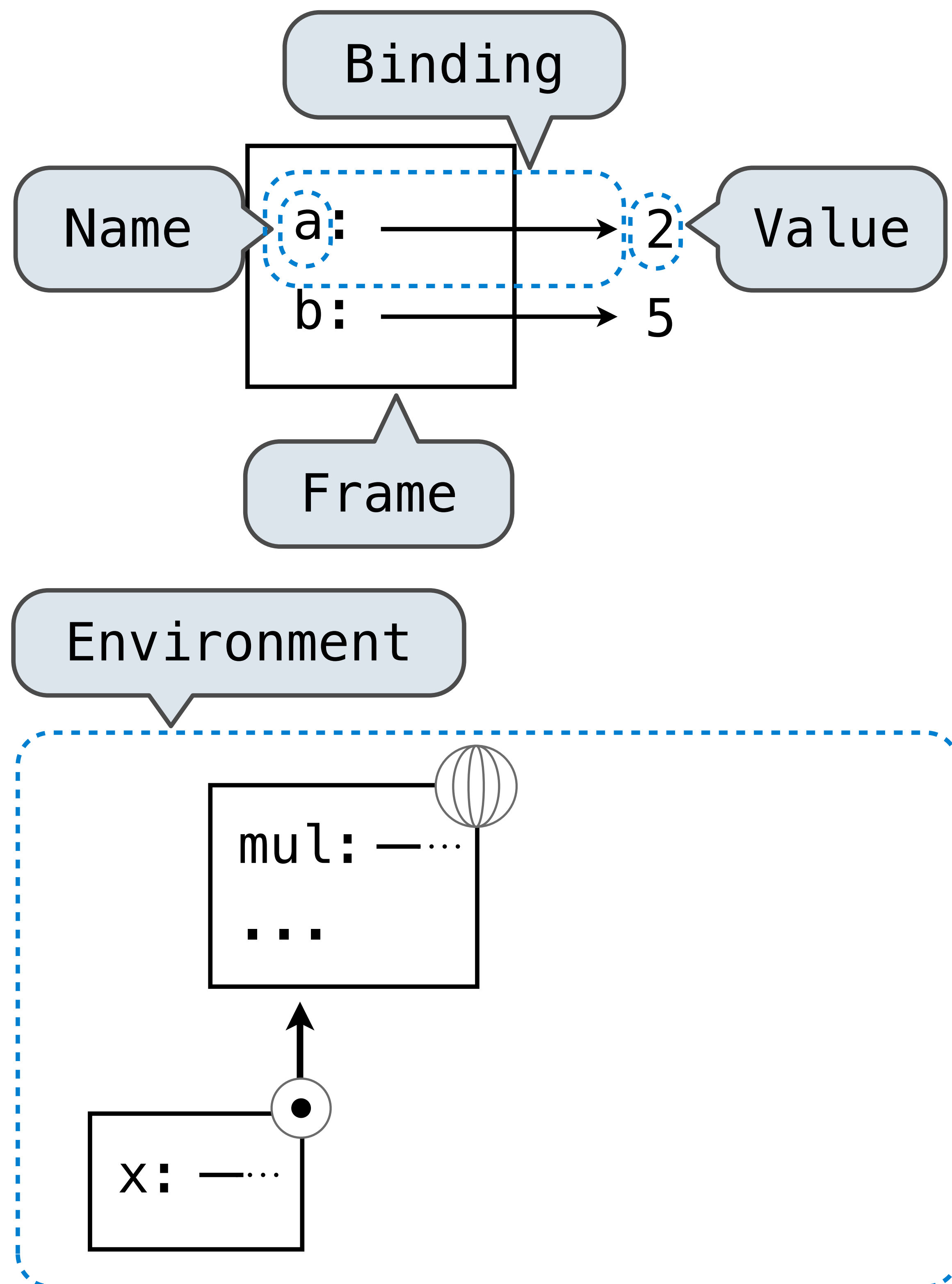
A frame is a rectangle that contains bindings

In a frame, there is at most one binding per name

Environments:

An environment is a ***sequence of frames***

Cast of Characters: Environment Diagrams



Frames:

A name is bound to a value

A frame is a rectangle that contains bindings

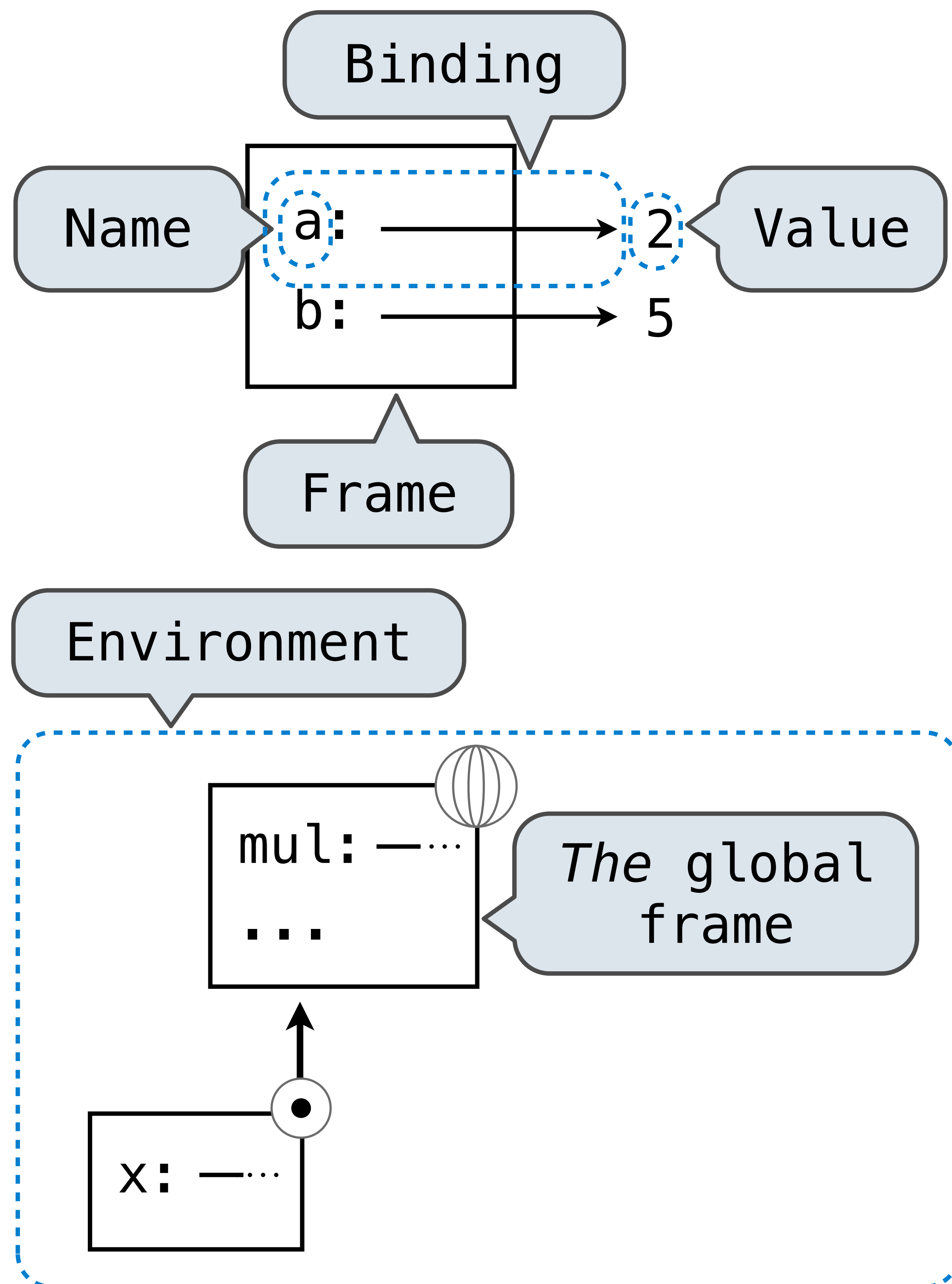
In a frame, there is at most one binding per name

Environments:

An environment is a ***sequence of frames***

So far, environments only have at most two frames

Cast of Characters: Environment Diagrams



Frames:

A name is bound to a value

A frame is a rectangle that contains bindings

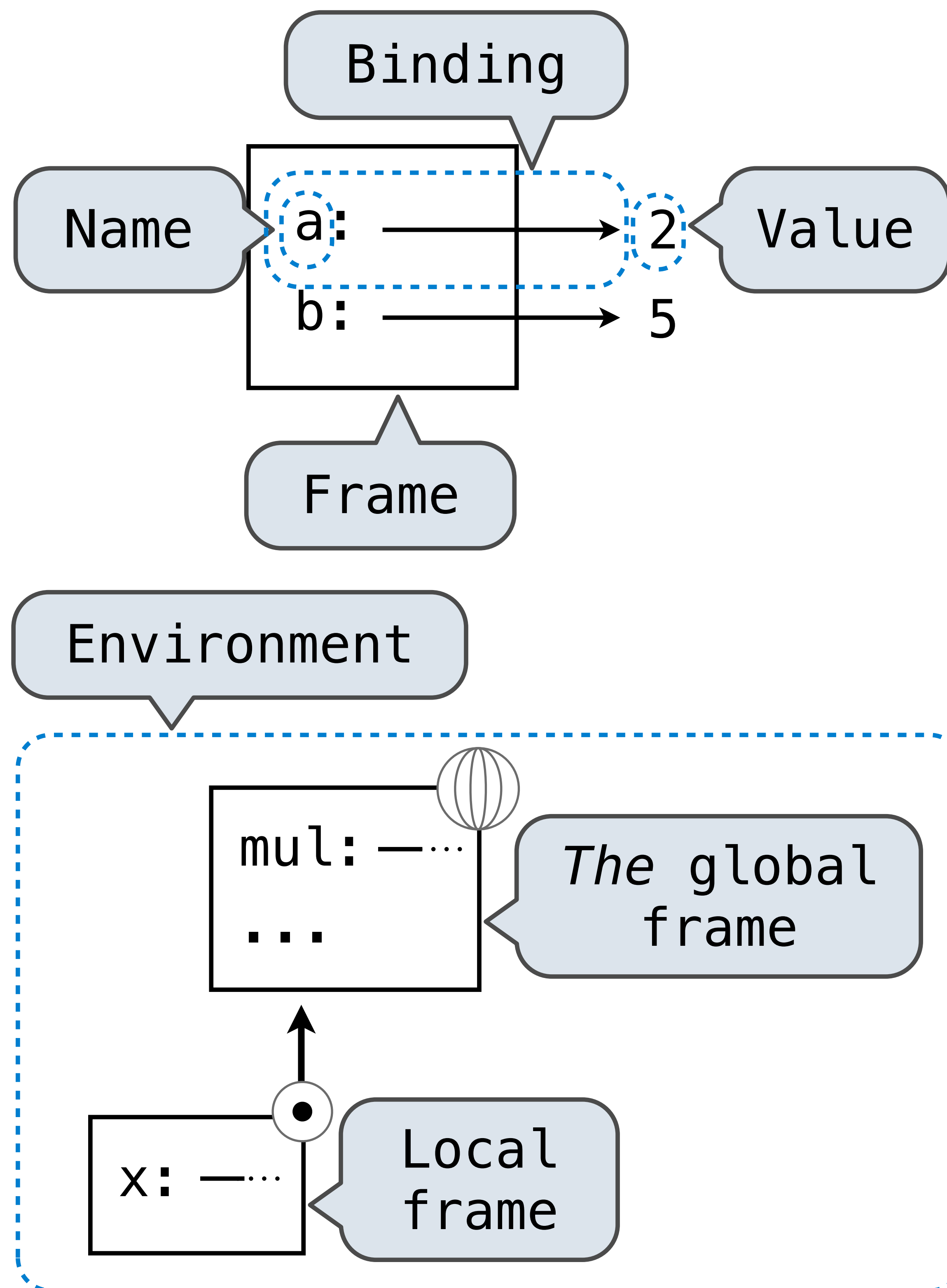
In a frame, there is at most one binding per name

Environments:

An environment is a ***sequence of frames***

So far, environments only have at most two frames

Cast of Characters: Environment Diagrams



Frames:

A name is bound to a value

A frame is a rectangle that contains bindings

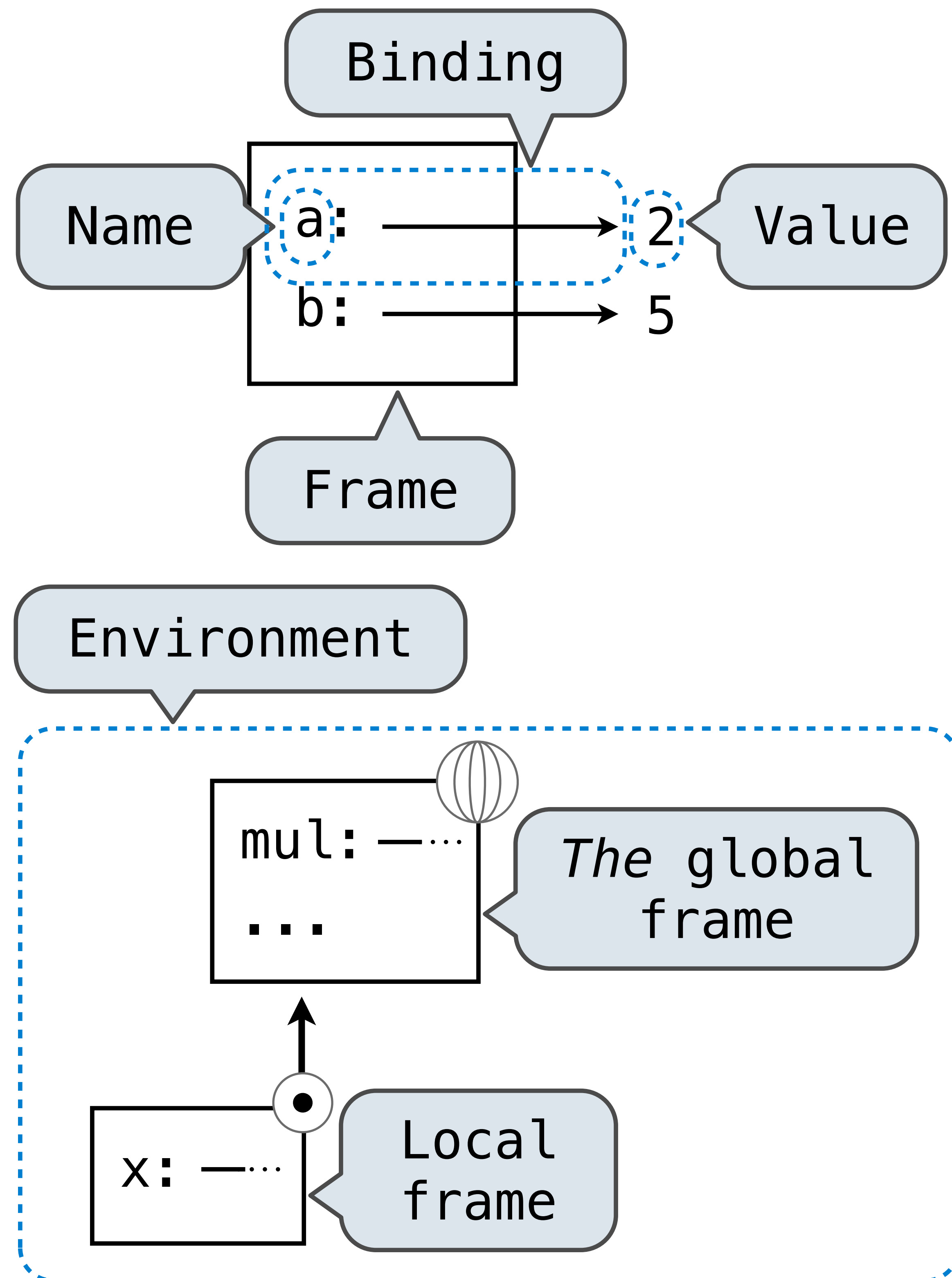
In a frame, there is at most one binding per name

Environments:

An environment is a ***sequence of frames***

So far, environments only have at most two frames

Cast of Characters: Environment Diagrams



Frames:

A name is bound to a value

A frame is a rectangle that contains bindings

In a frame, there is at most one binding per name

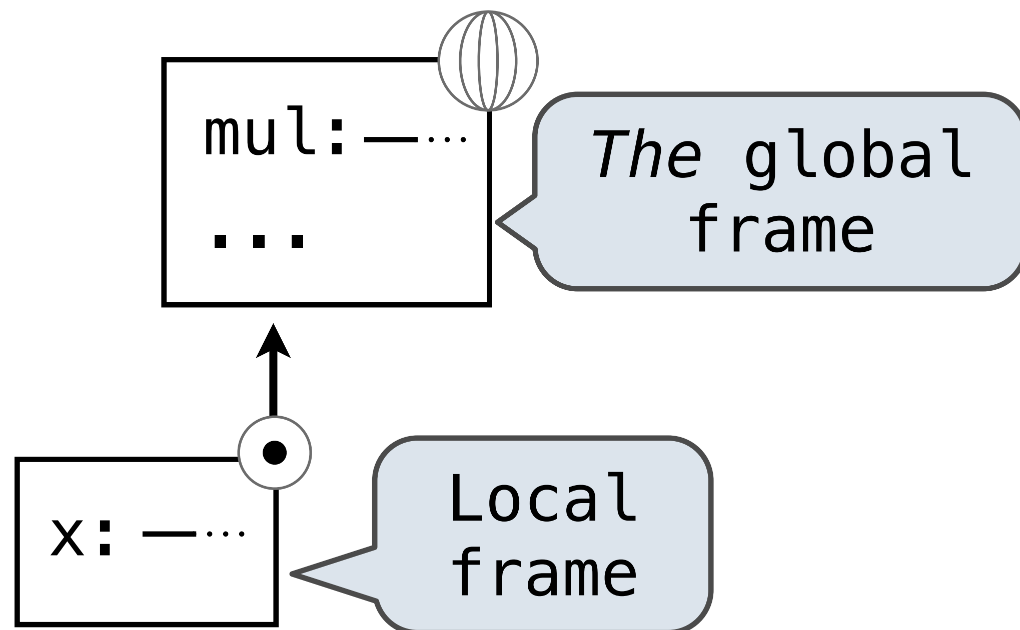
Environments:

An environment is a ***sequence of frames***

So far, environments only have at most two frames

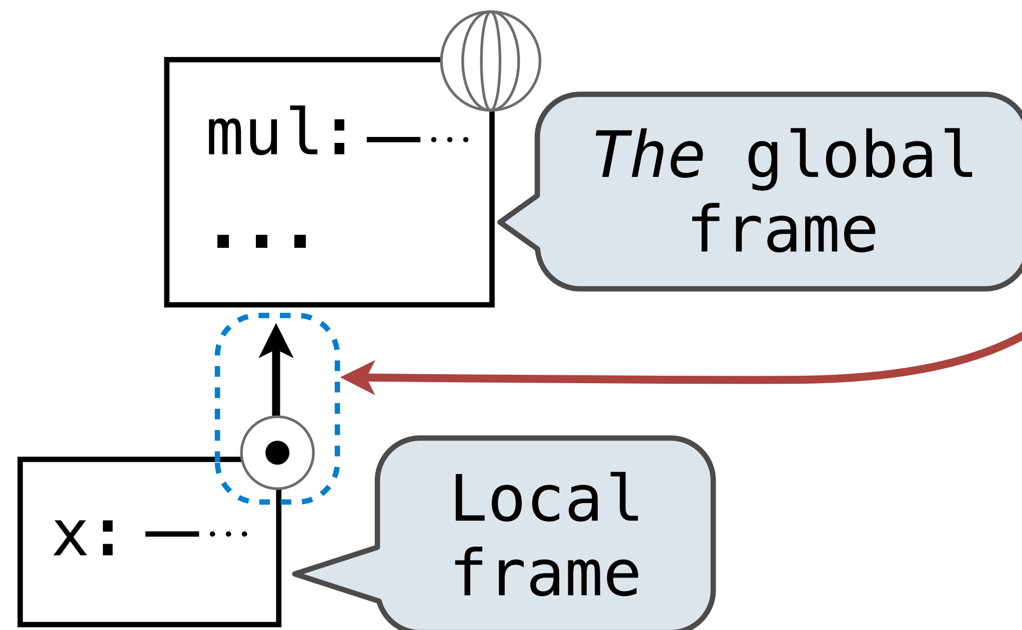
(Friday: longer sequences)

An Environment is a Sequence of Frames



Environments (Memory):

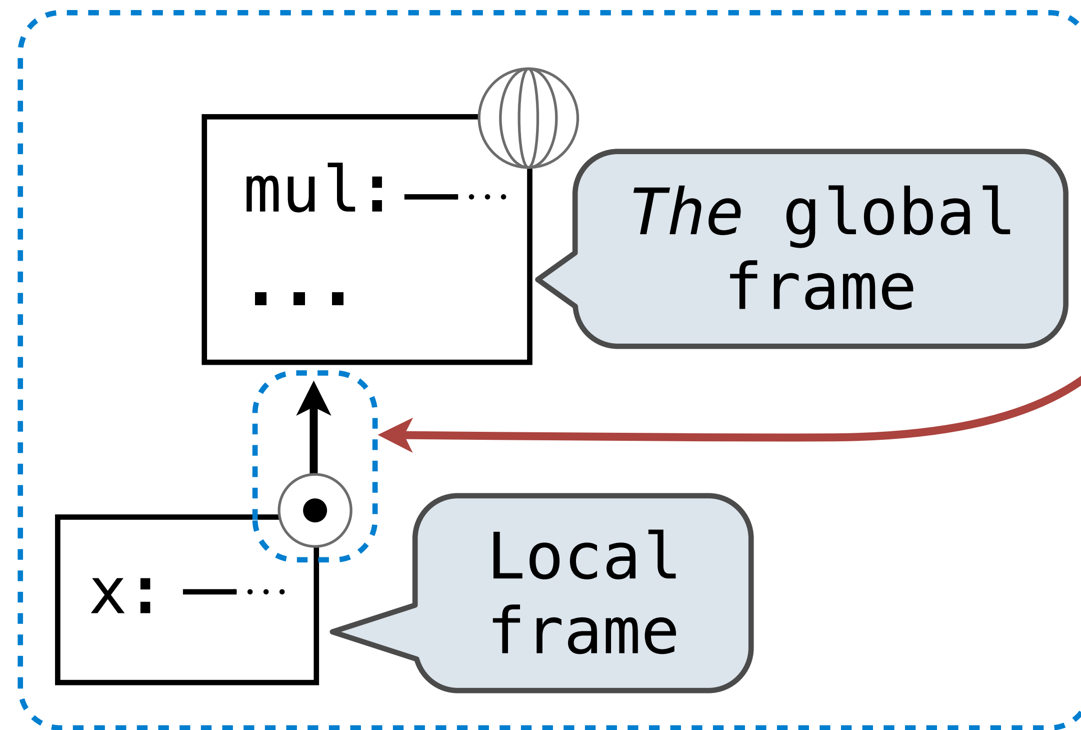
An Environment is a Sequence of Frames



Environments (Memory):

Frames link to each other

An Environment is a Sequence of Frames

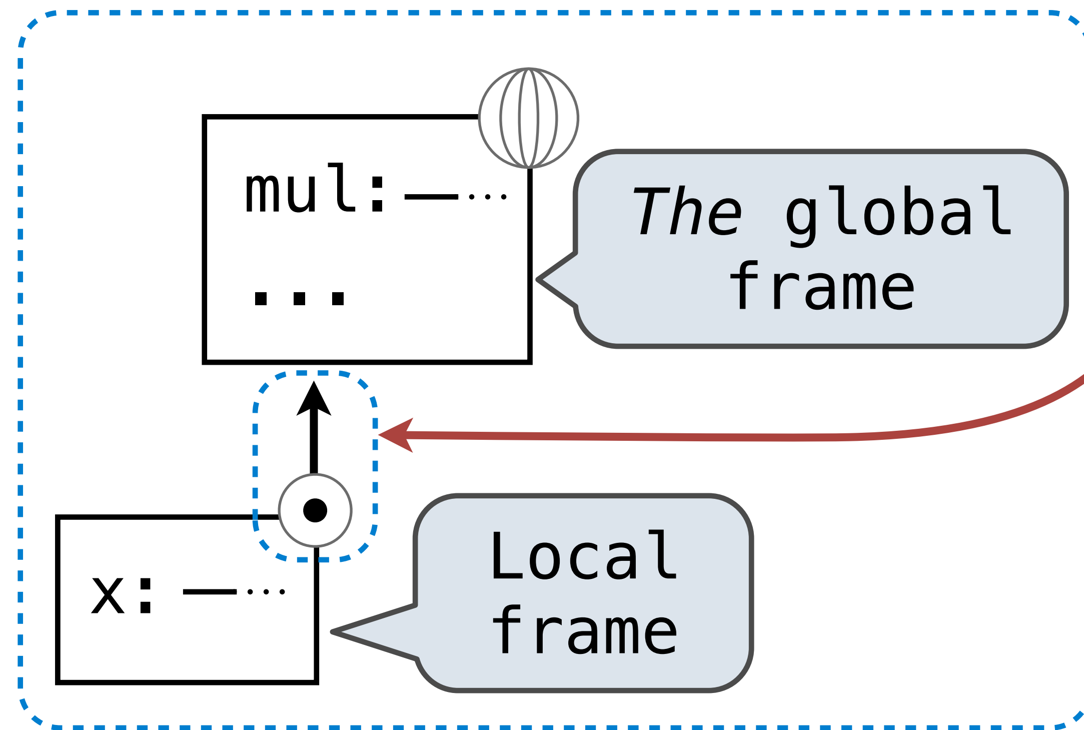


Environments (Memory):

Frames link to each other

An environment is a ***sequence of frames***

An Environment is a Sequence of Frames



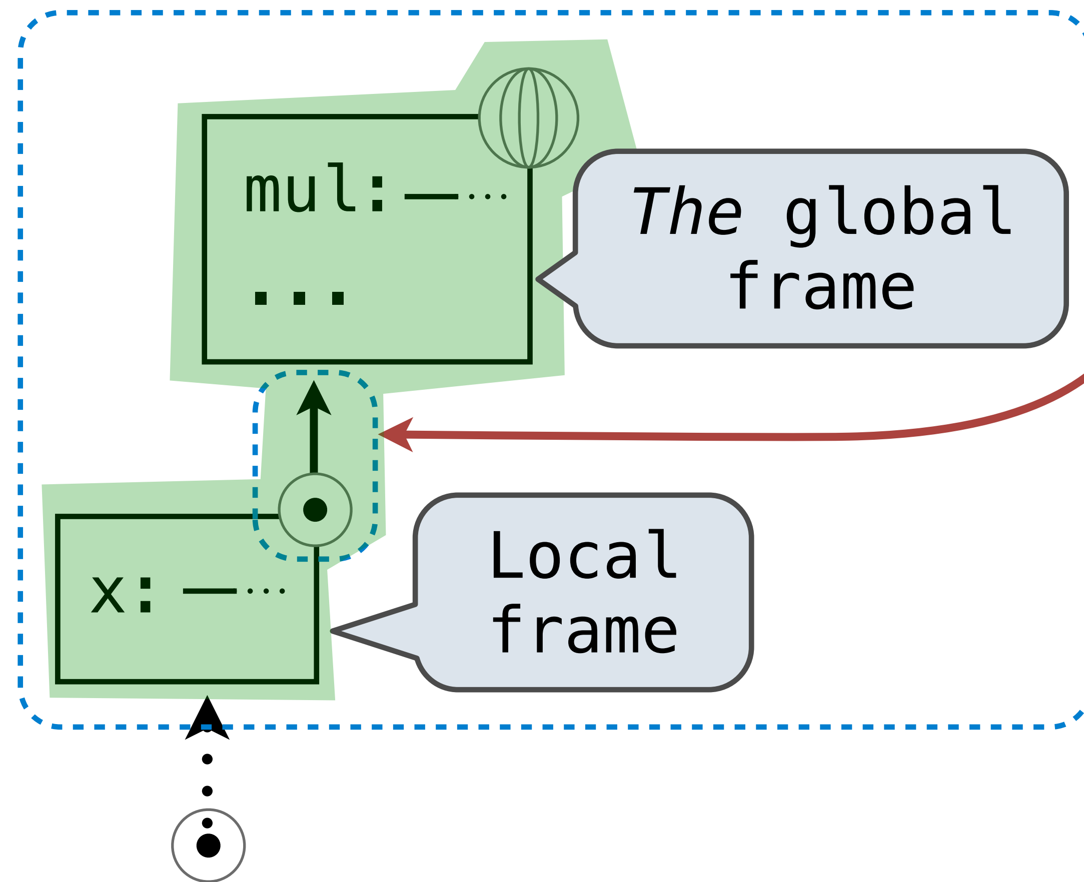
Environments (Memory):

Frames link to each other

An environment is a ***sequence of frames***

An environment is a first frame, plus the frames that follow

An Environment is a Sequence of Frames



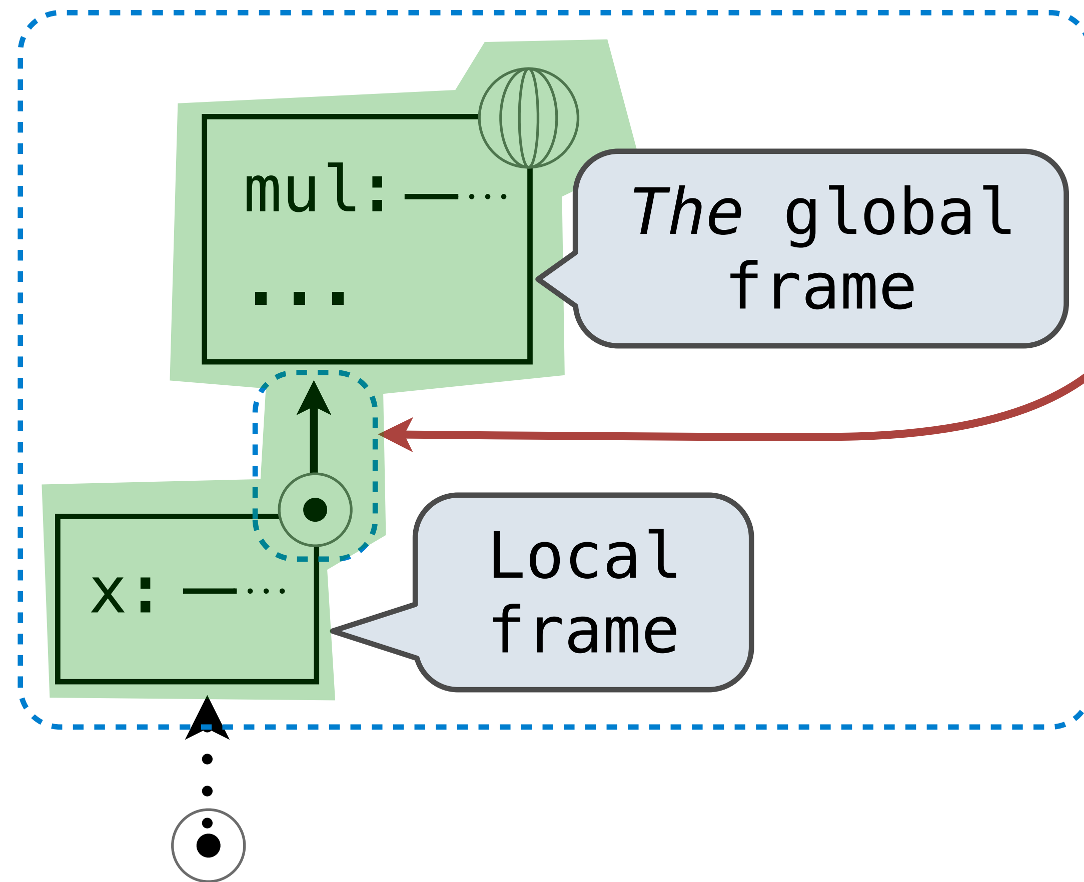
Environments (Memory):

Frames link to each other

An environment is a ***sequence of frames***

An environment is a first frame, plus the frames that follow

An Environment is a Sequence of Frames



Environments (Memory):

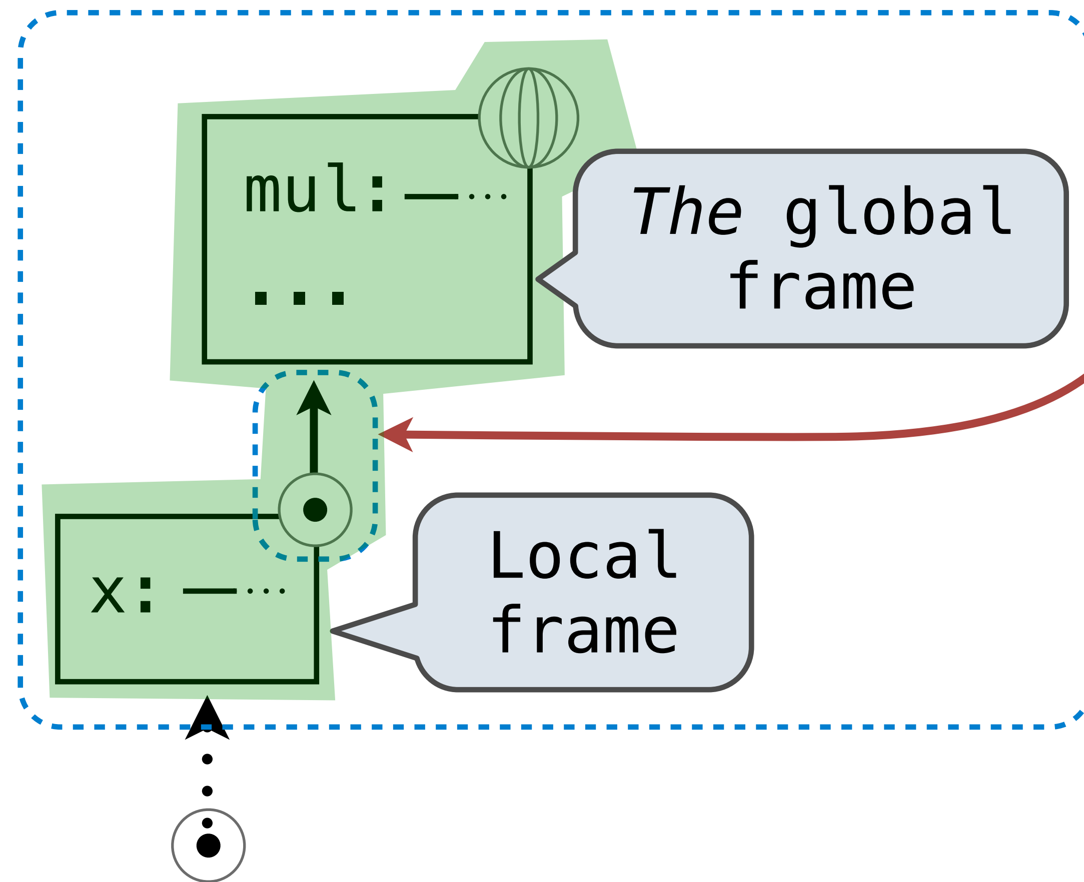
Frames link to each other

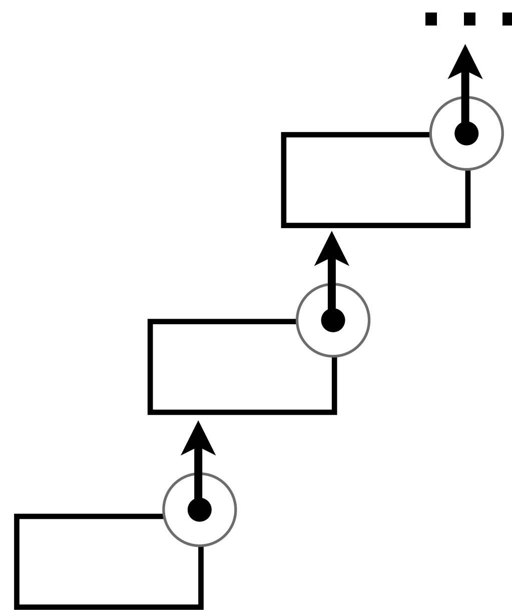
An environment is a ***sequence of frames***

An environment is a first frame, plus the frames that follow

An environment is a first frame, plus the ***sequence of frames*** that follow

An Environment is a Sequence of Frames





Environments (Memory):

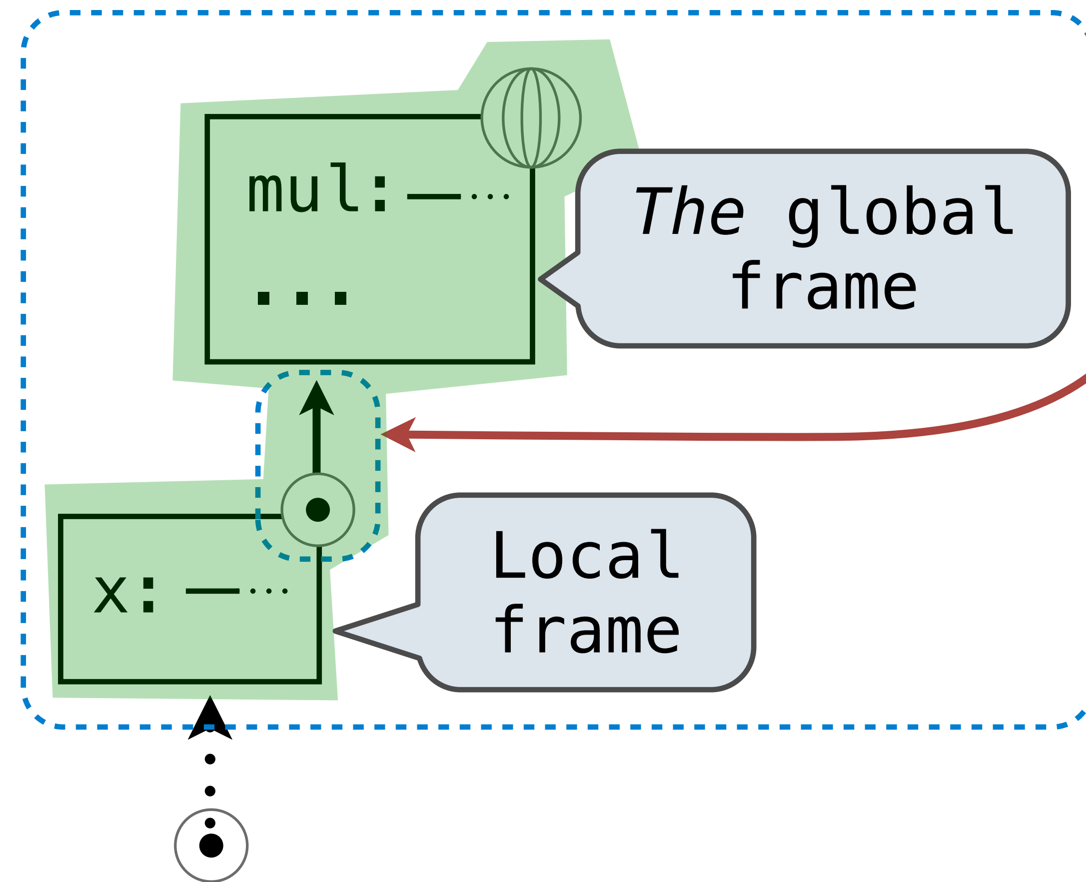
Frames link to each other

An environment is a ***sequence of frames***

An environment is a first frame, plus the frames that follow

An environment is a first frame, plus the ***sequence of frames*** that follow

An Environment is a Sequence of Frames



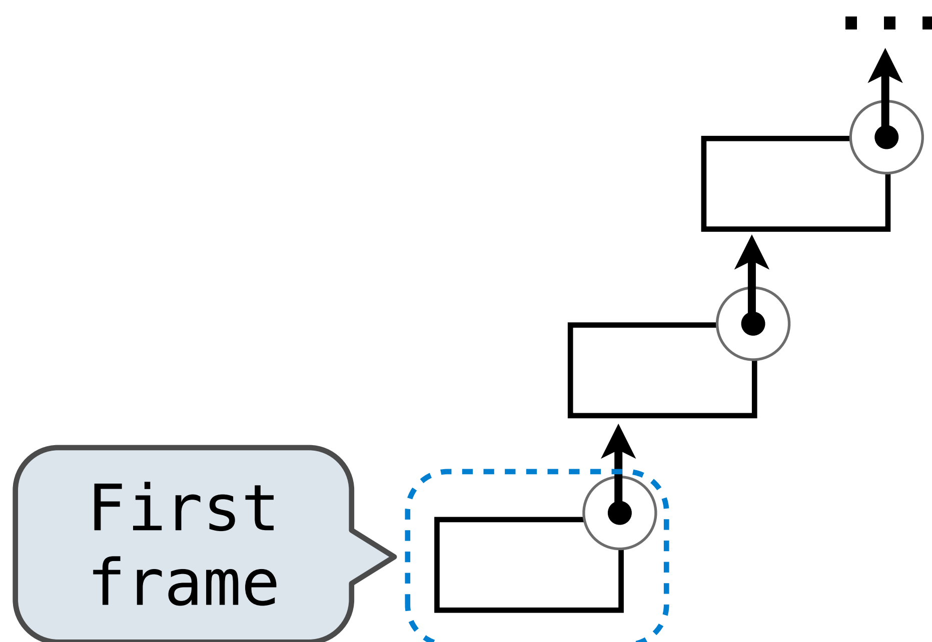
Environments (Memory):

Frames link to each other

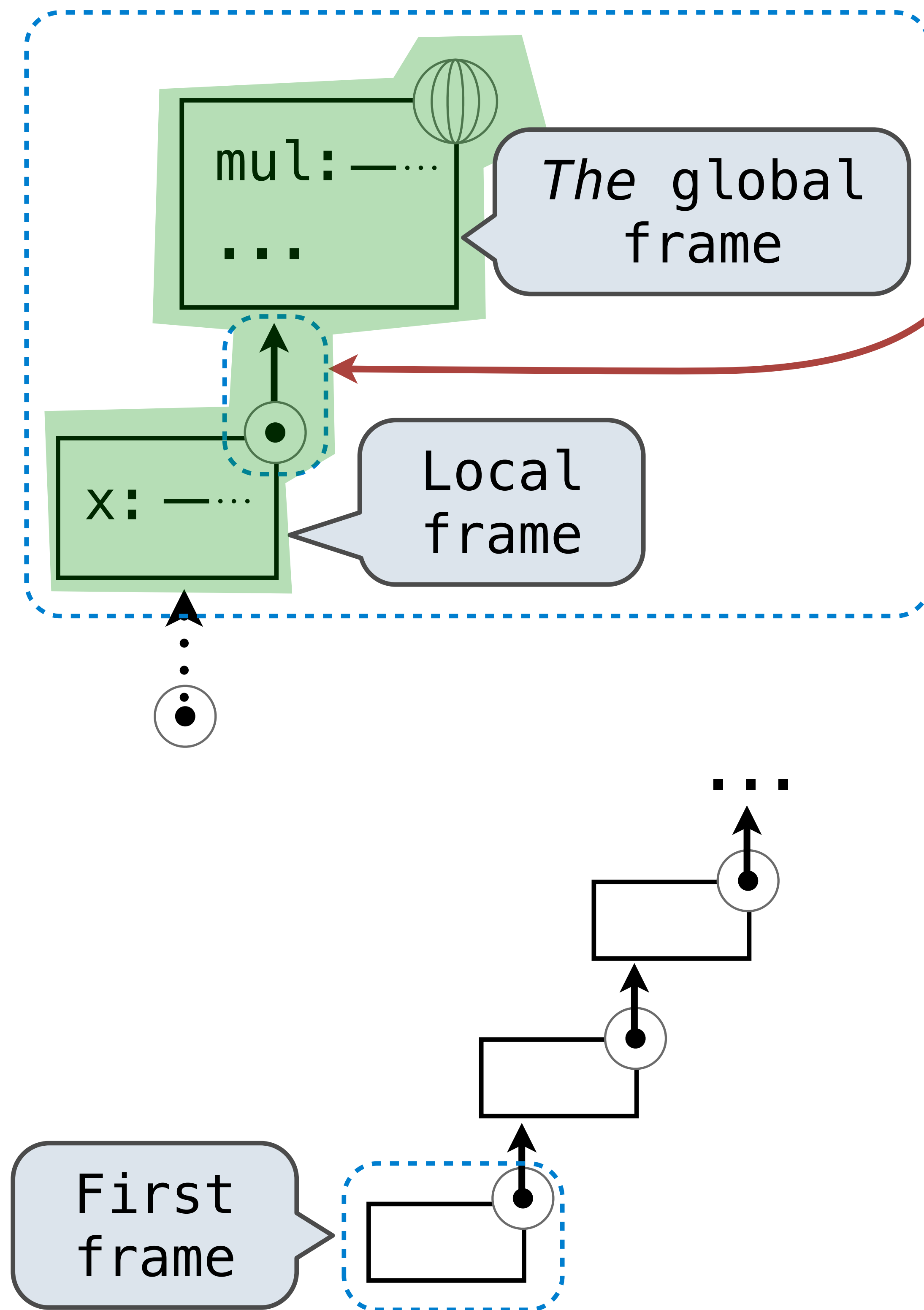
An environment is a ***sequence of frames***

An environment is a first frame, plus the frames that follow

An environment is a first frame, plus the ***sequence of frames*** that follow



An Environment is a Sequence of Frames



Environments (Memory):

Frames link to each other

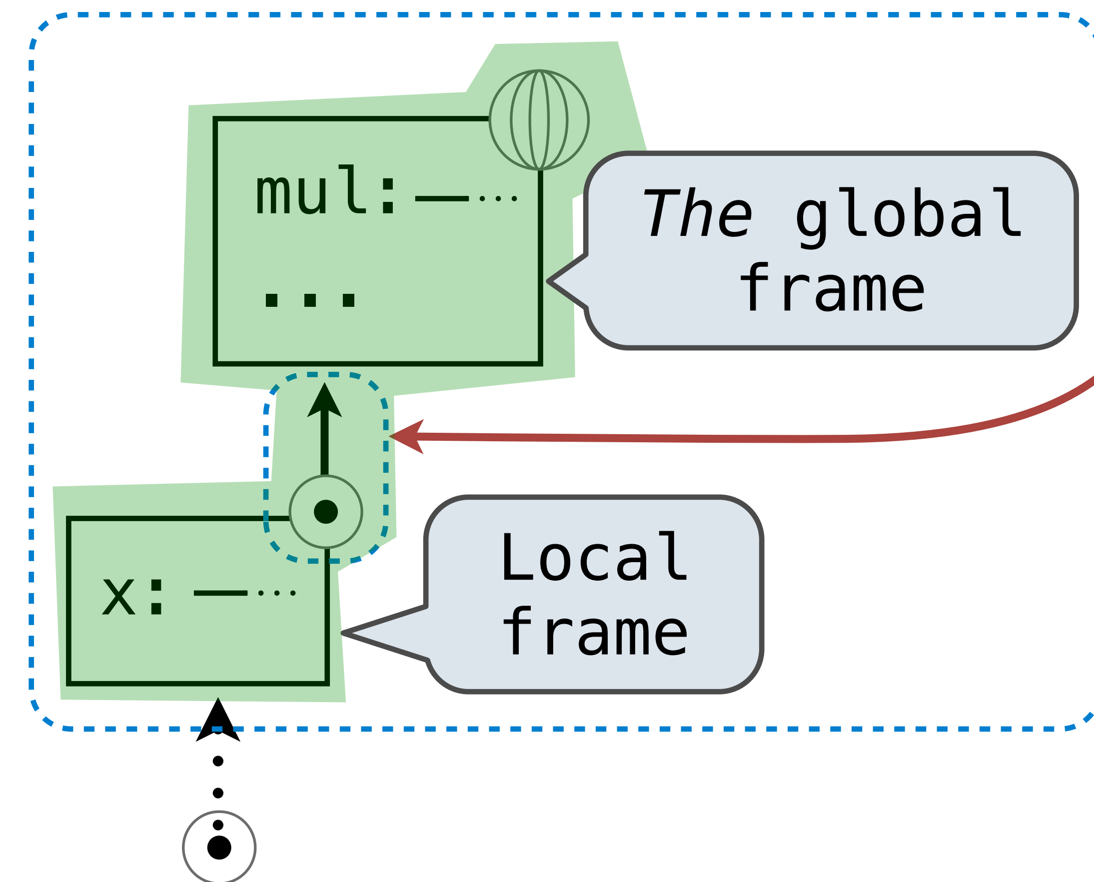
An environment is a ***sequence of frames***

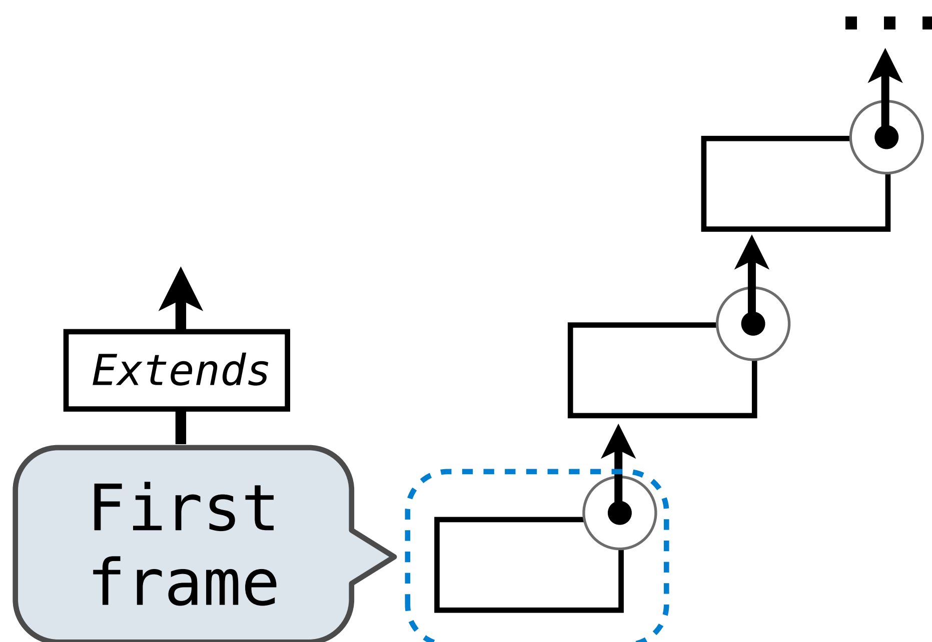
An environment is a first frame, plus the frames that follow

An environment is a first frame, plus the ***sequence of frames*** that follow

An environment is a first frame, plus the **environment** that follows

An Environment is a Sequence of Frames





Environments (Memory):

Frames link to each other

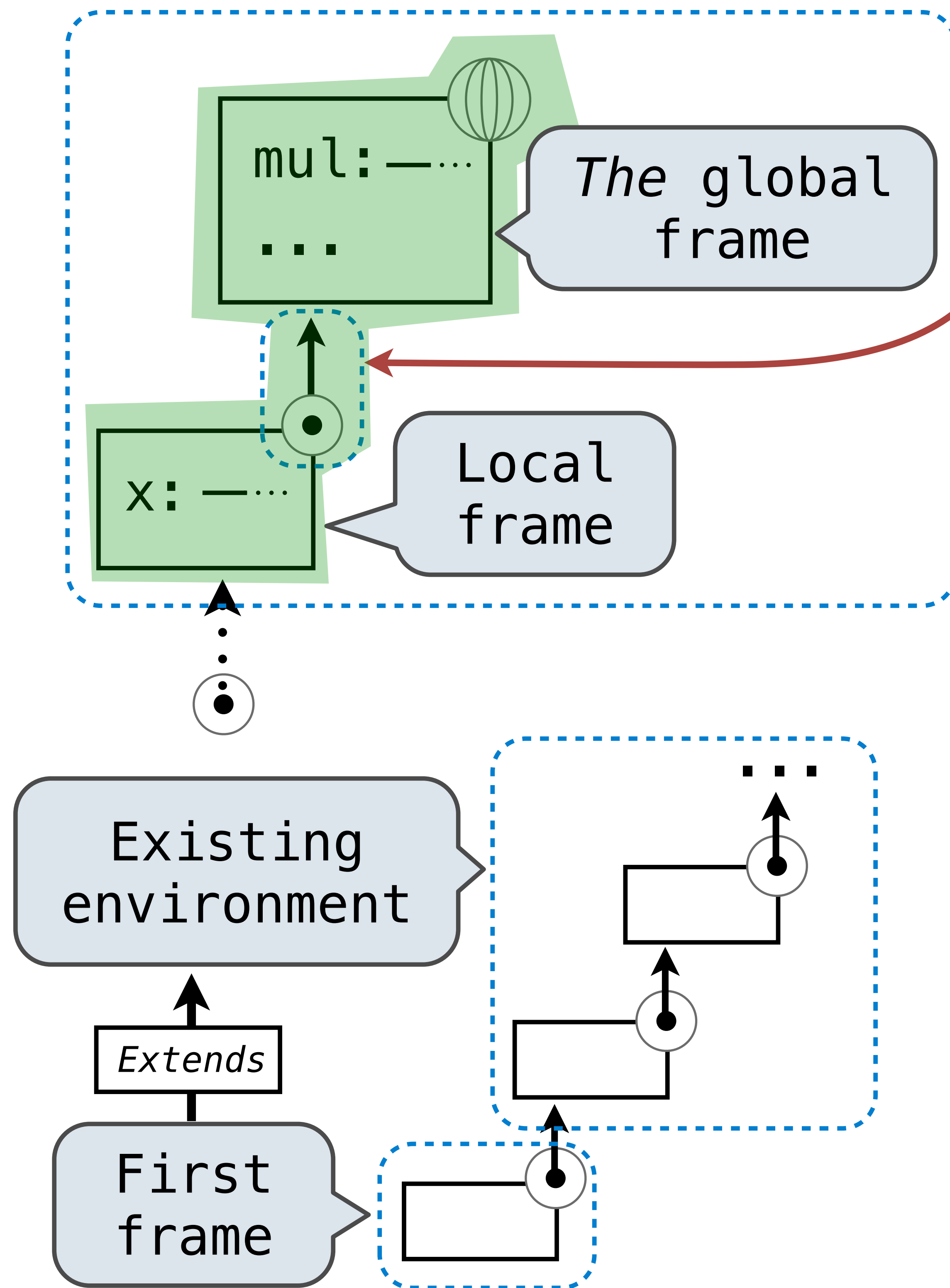
An environment is a ***sequence of frames***

An environment is a first frame, plus the frames that follow

An environment is a first frame, plus the ***sequence of frames*** that follow

An environment is a first frame, plus the **environment** that follows

An Environment is a Sequence of Frames



Environments (Memory):

Frames link to each other

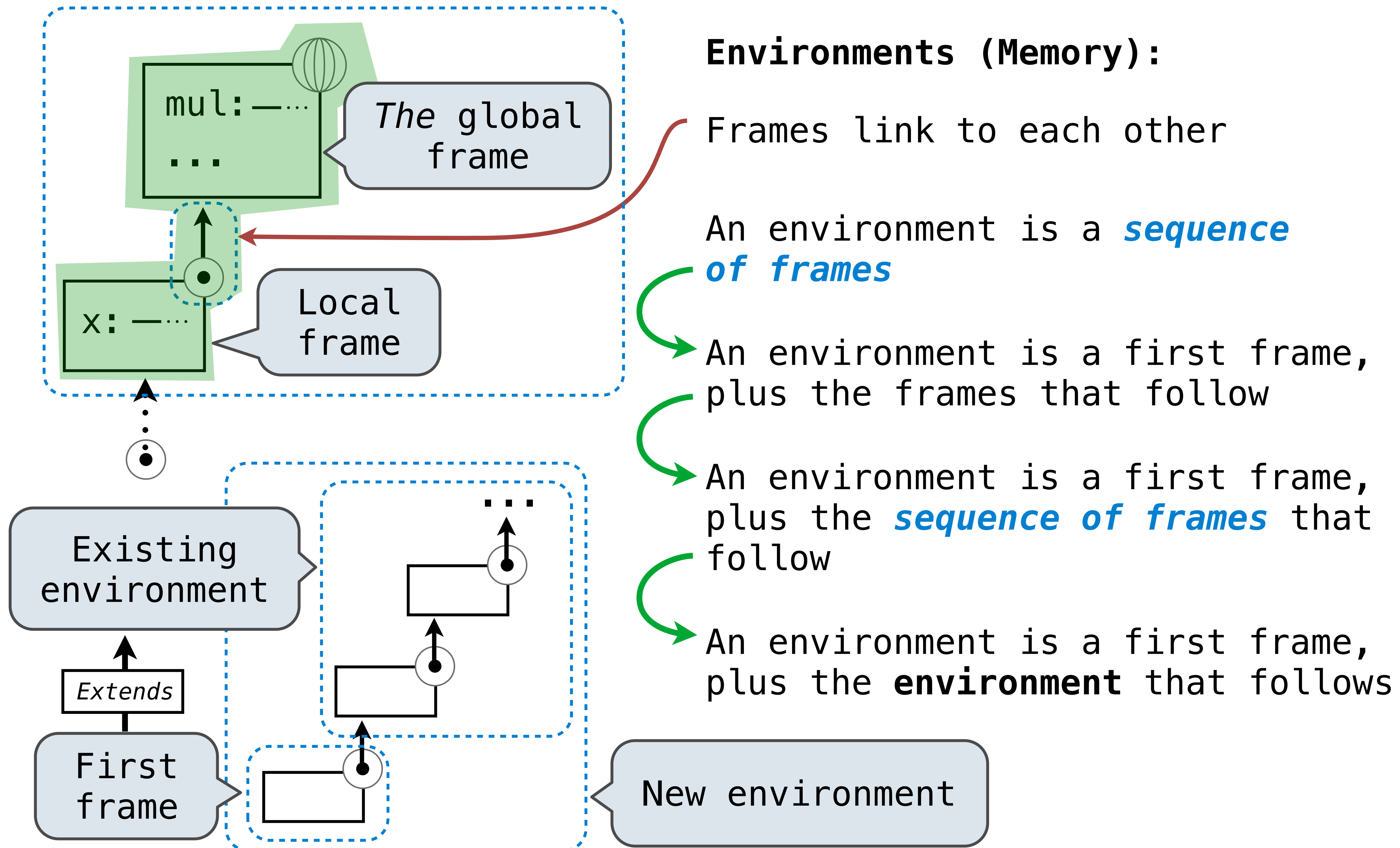
An environment is a ***sequence of frames***

An environment is a first frame, plus the frames that follow

An environment is a first frame, plus the ***sequence of frames*** that follow

An environment is a first frame, plus the **environment** that follows

An Environment is a Sequence of Frames



Environments (Memory):

Frames link to each other

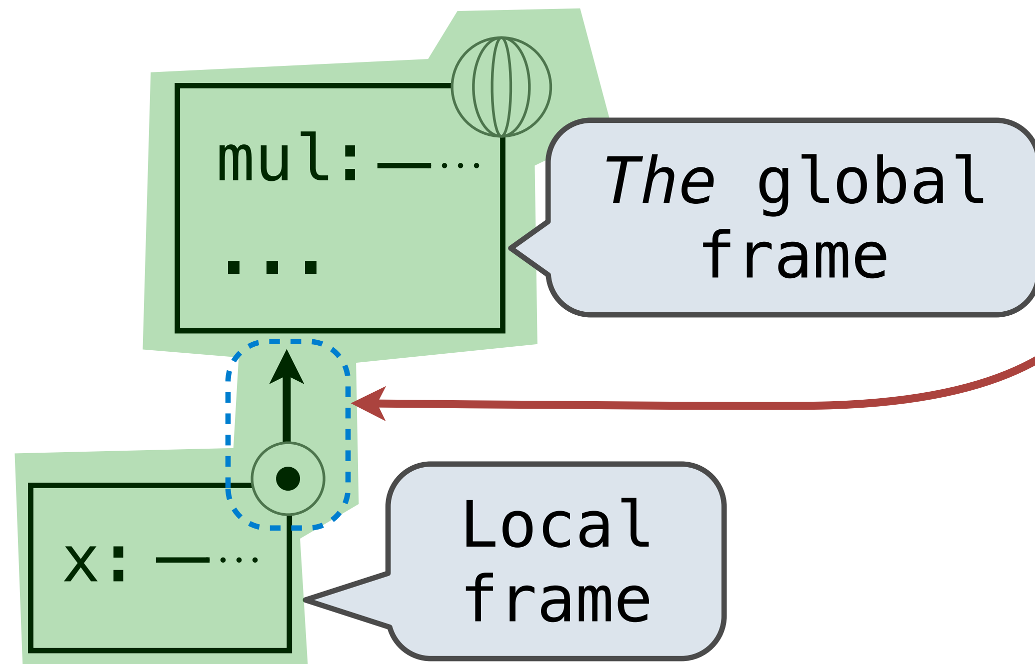
An environment is a ***sequence of frames***

An environment is a first frame, plus the frames that follow

An environment is a first frame, plus the ***sequence of frames*** that follow

An environment is a first frame, plus the **environment** that follows

An Expression is Evaluated in an Environment



Environments (Memory):

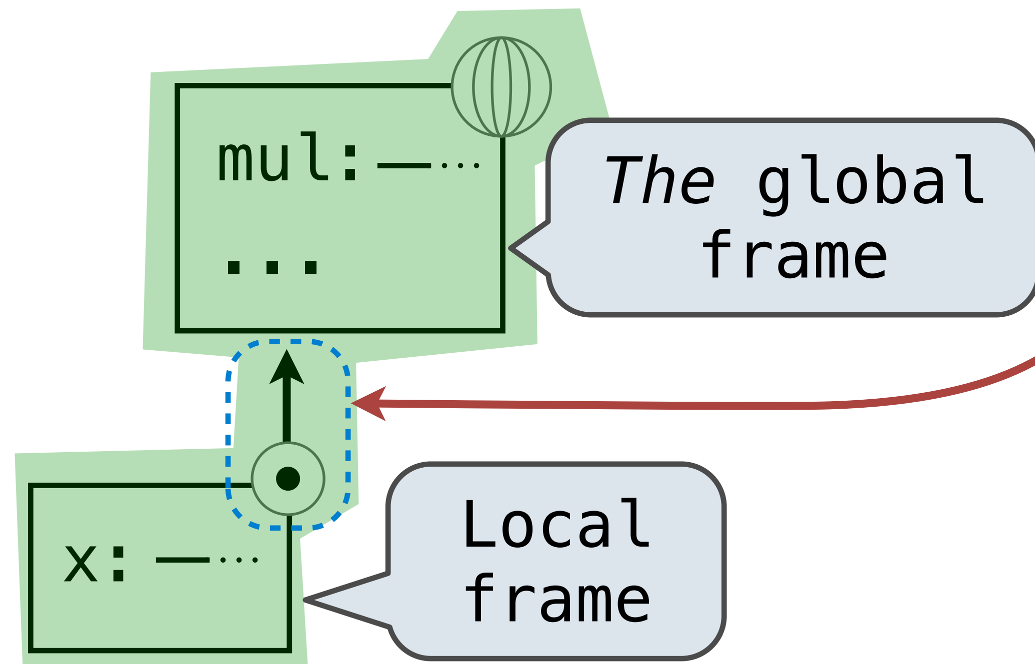
Frames link to each other

An environment is a ***sequence*** of frames

An environment is a first frame, plus the **environment** that follows

Expressions (Program):

An Expression is Evaluated in an Environment



Environments (Memory):

Frames link to each other

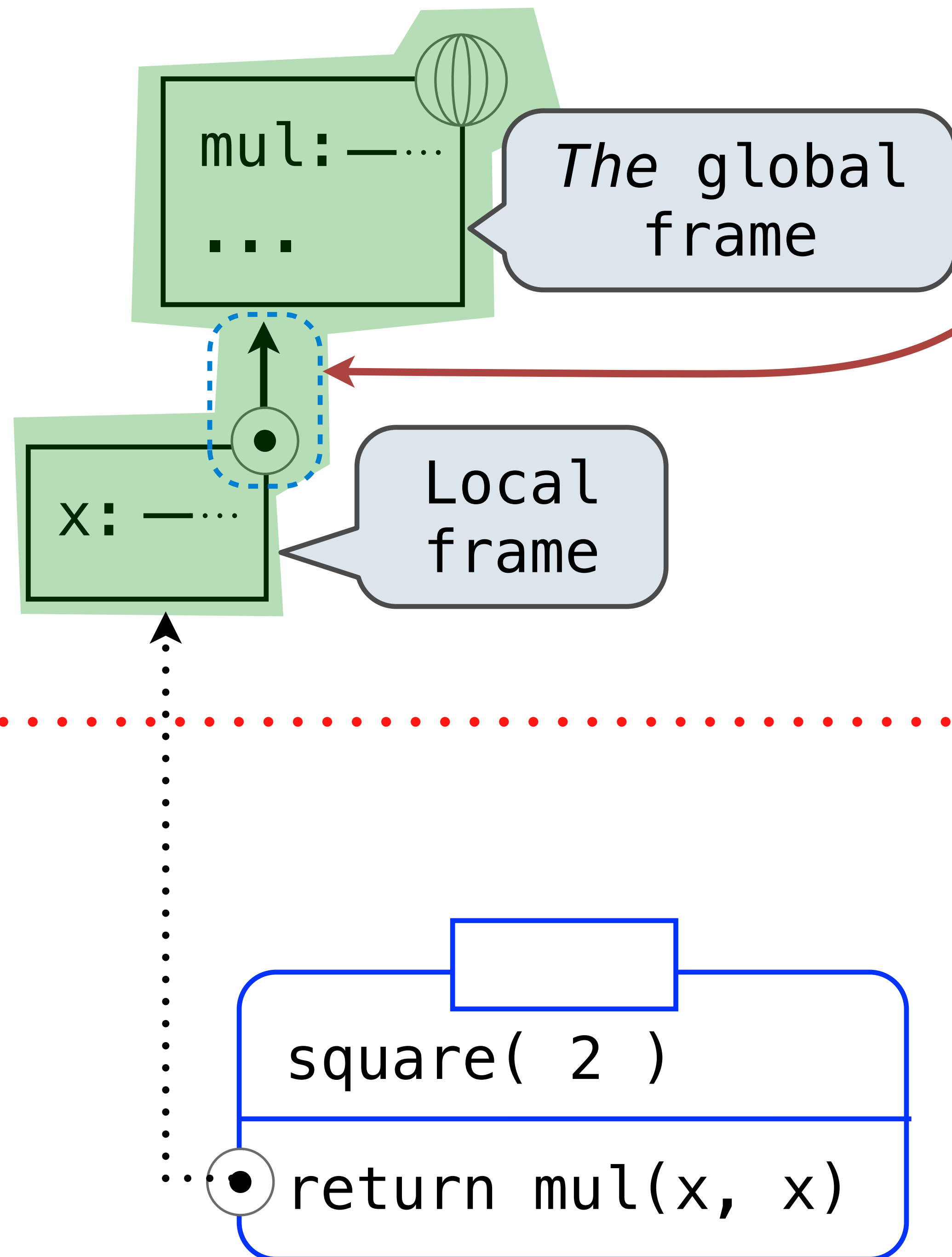
An environment is a ***sequence*** of frames

An environment is a first frame, plus the **environment** that follows

Expressions (Program):

Expressions are Python code

An Expression is Evaluated in an Environment



Environments (Memory):

Frames link to each other

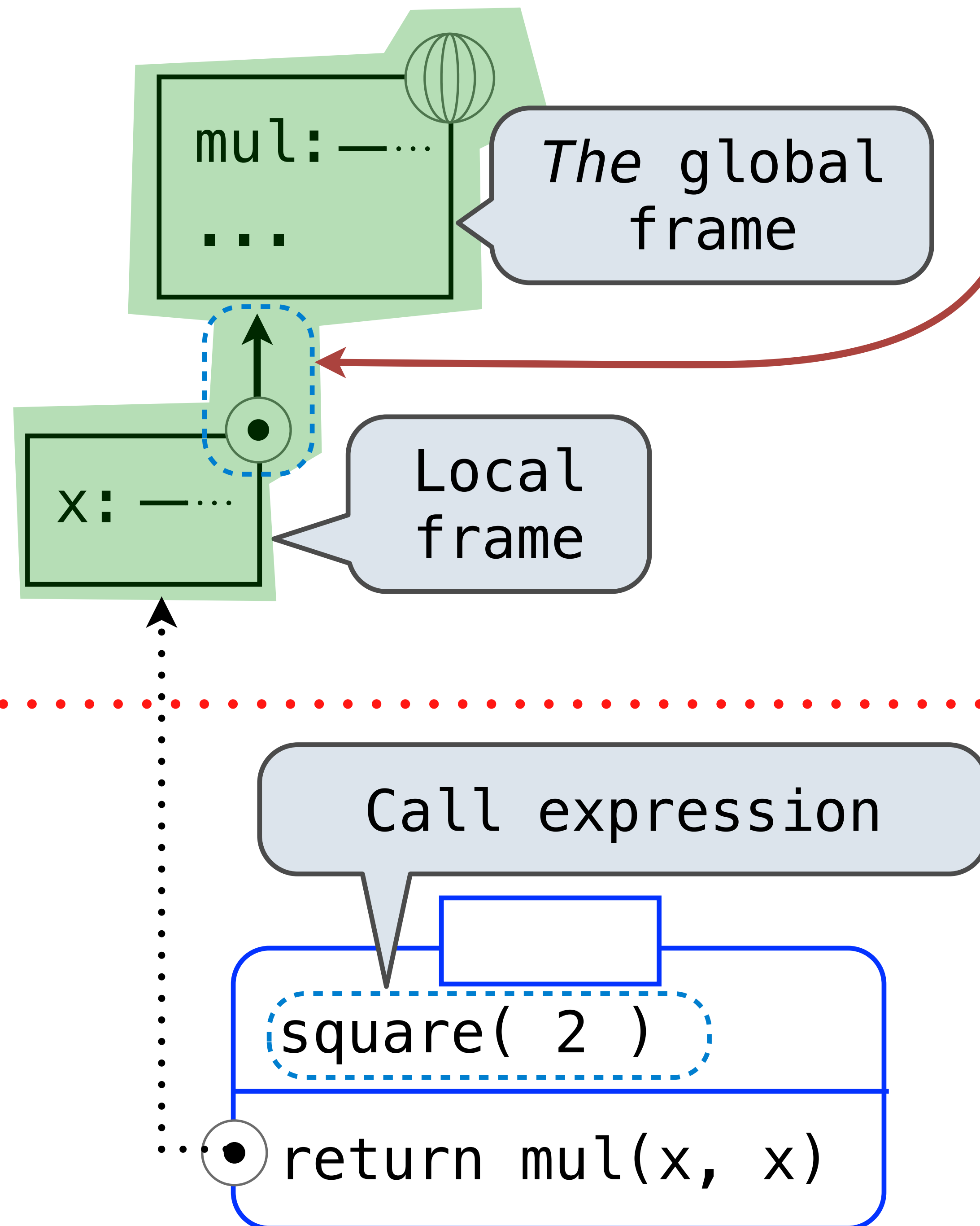
An environment is a ***sequence*** of frames

An environment is a first frame, plus the **environment** that follows

Expressions (Program):

Expressions are Python code

An Expression is Evaluated in an Environment



Environments (Memory):

Frames link to each other

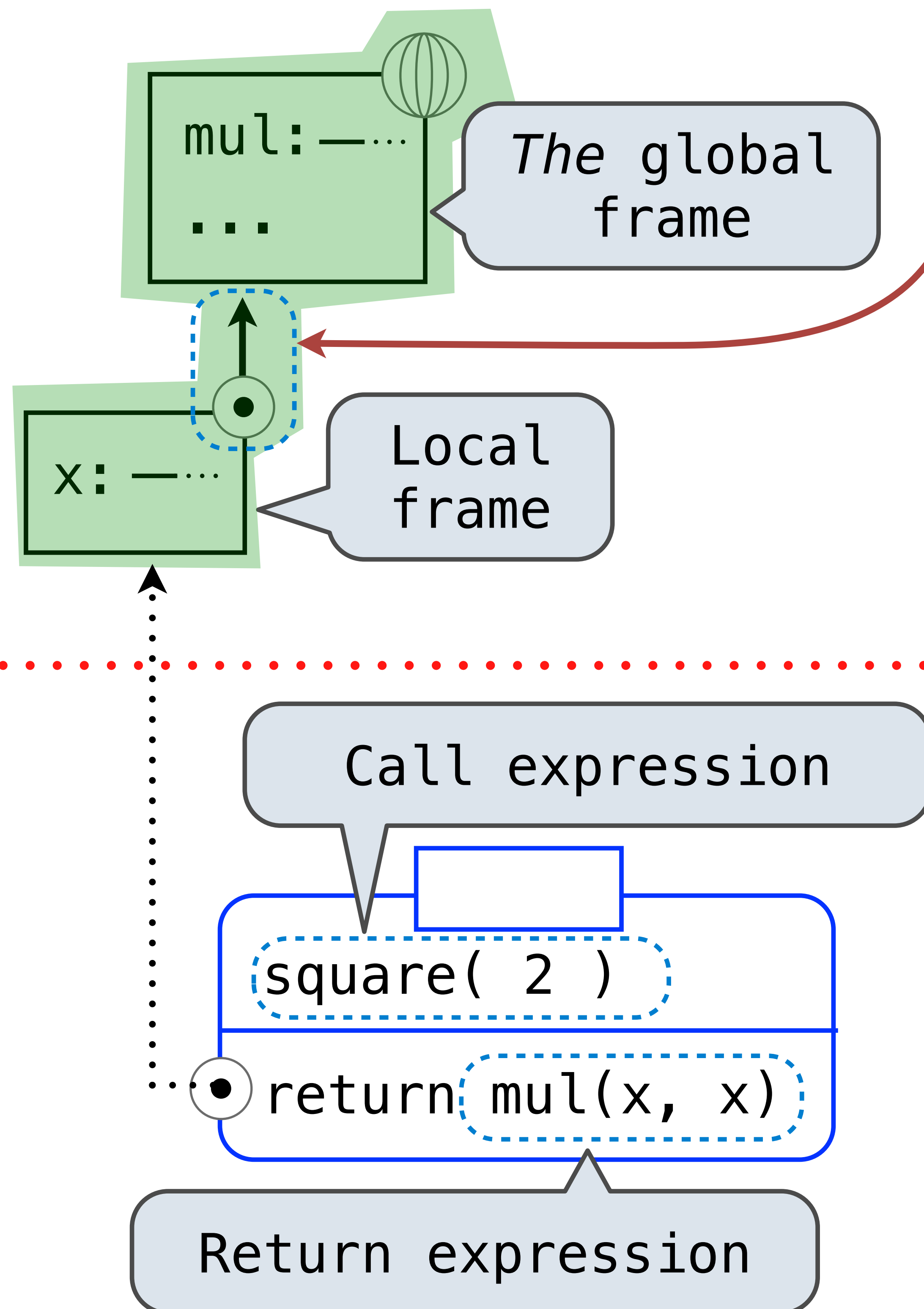
An environment is a ***sequence*** of frames

An environment is a first frame, plus the **environment** that follows

Expressions (Program):

Expressions are Python code

An Expression is Evaluated in an Environment



Environments (Memory):

Frames link to each other

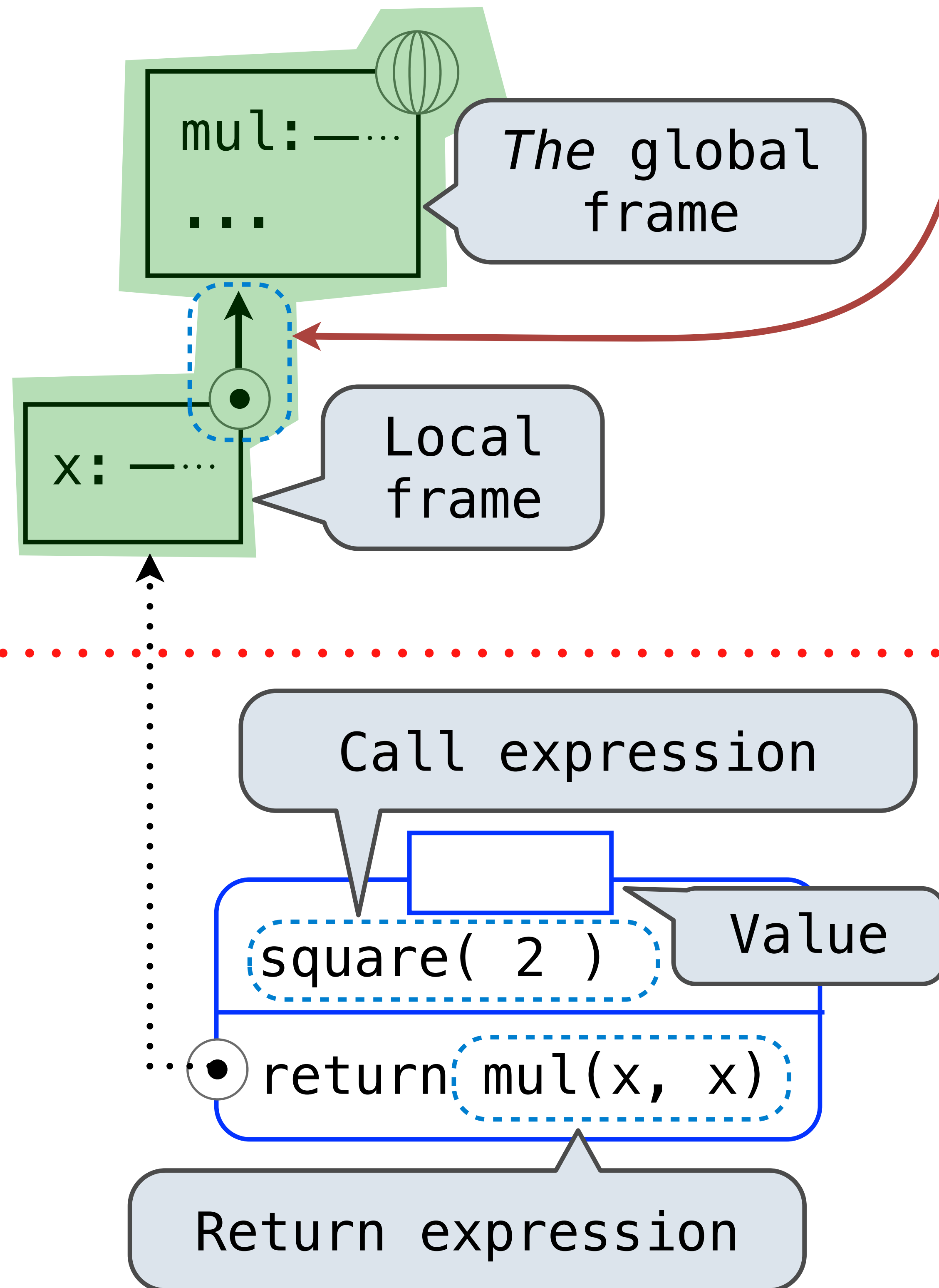
An environment is a ***sequence*** of frames

An environment is a first frame, plus the **environment** that follows

Expressions (Program):

Expressions are Python code

An Expression is Evaluated in an Environment



Environments (Memory):

Frames link to each other

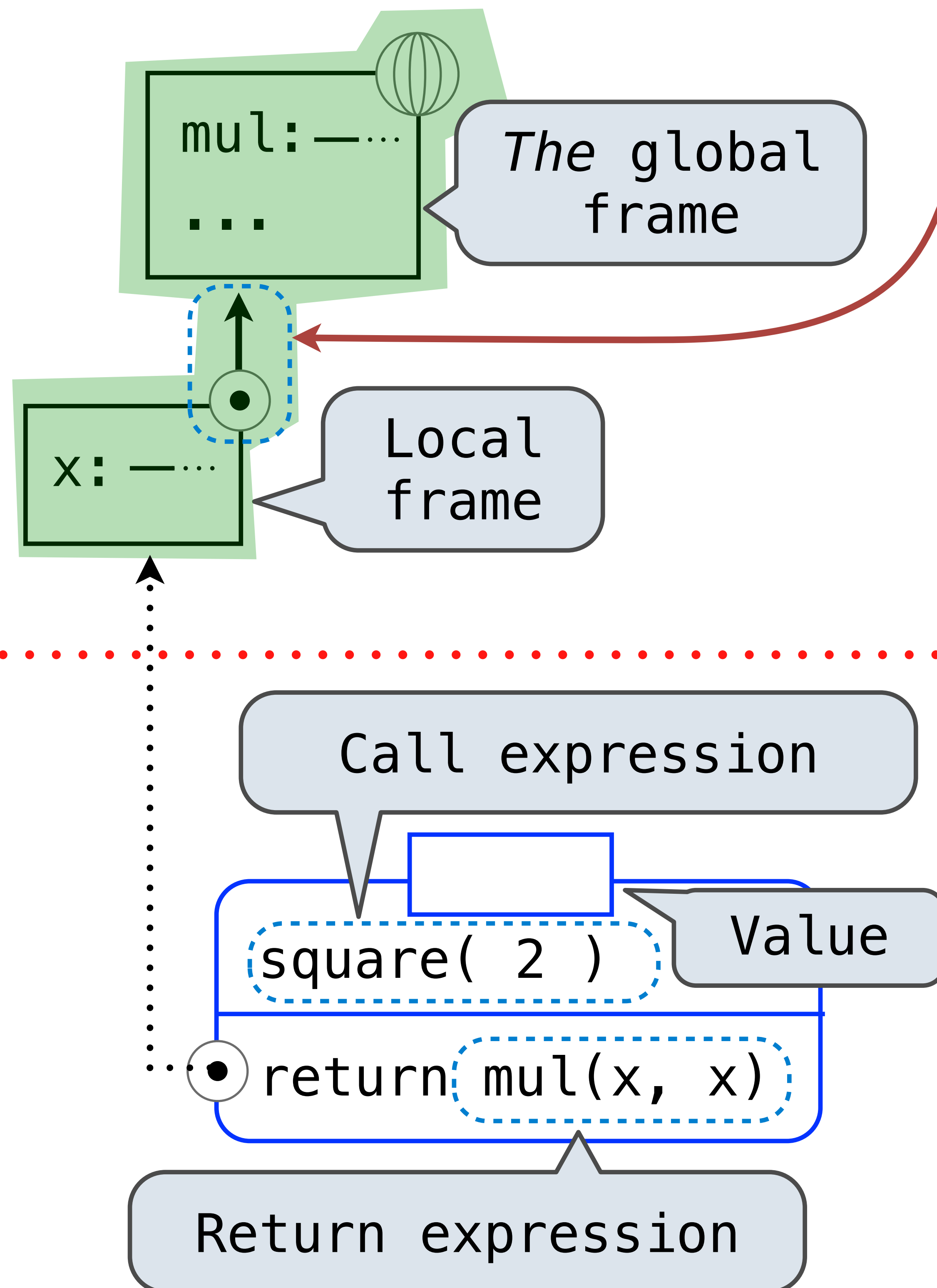
An environment is a ***sequence*** of frames

An environment is a first frame, plus the **environment** that follows

Expressions (Program):

Expressions are Python code

An Expression is Evaluated in an Environment



Environments (Memory):

Frames link to each other

An environment is a ***sequence*** of frames

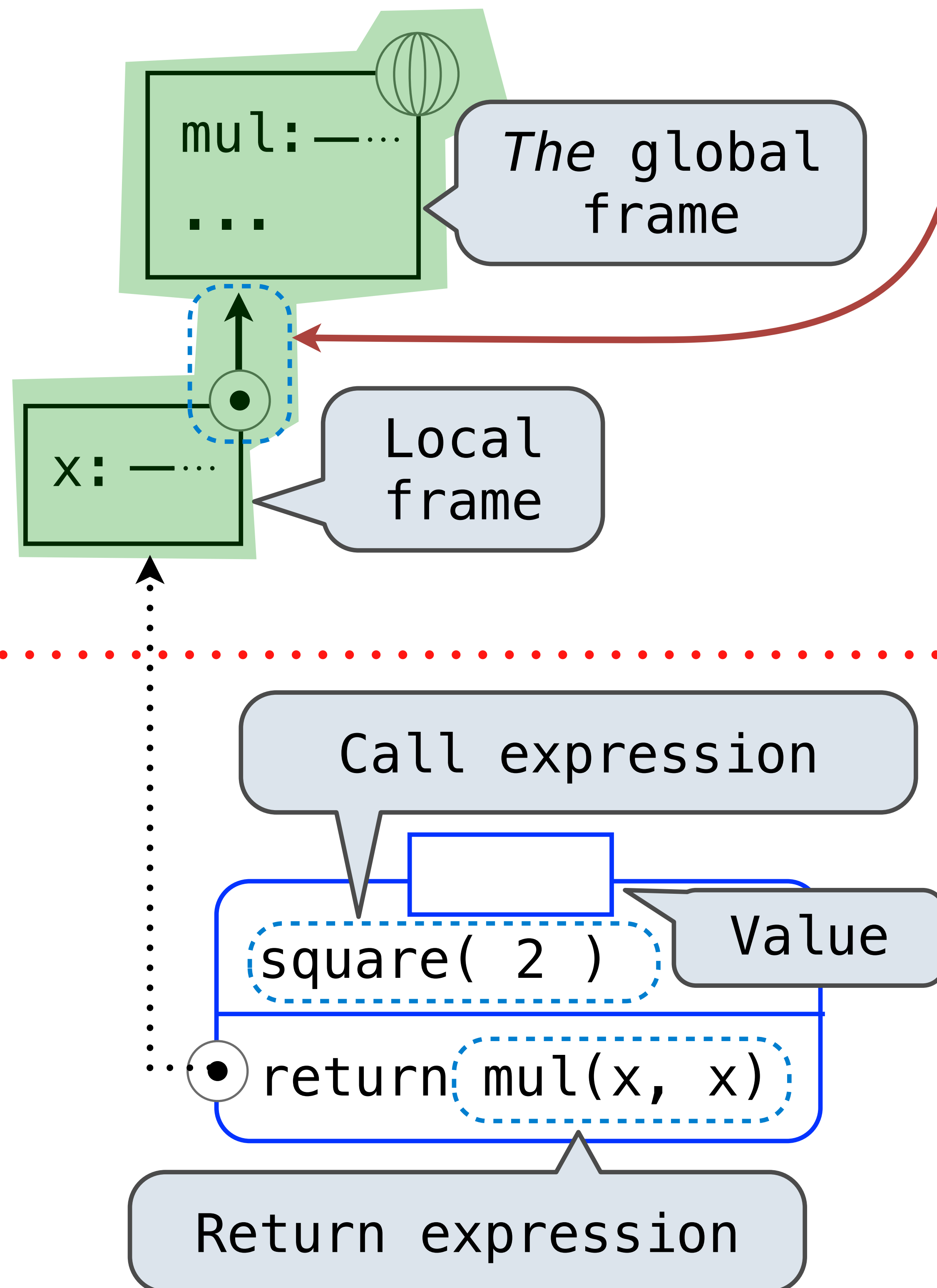
An environment is a first frame, plus the **environment** that follows

Expressions (Program):

Expressions are Python code

Not part of an environment

An Expression is Evaluated in an Environment



Environments (Memory):

Frames link to each other

An environment is a ***sequence*** of frames

An environment is a first frame, plus the **environment** that follows

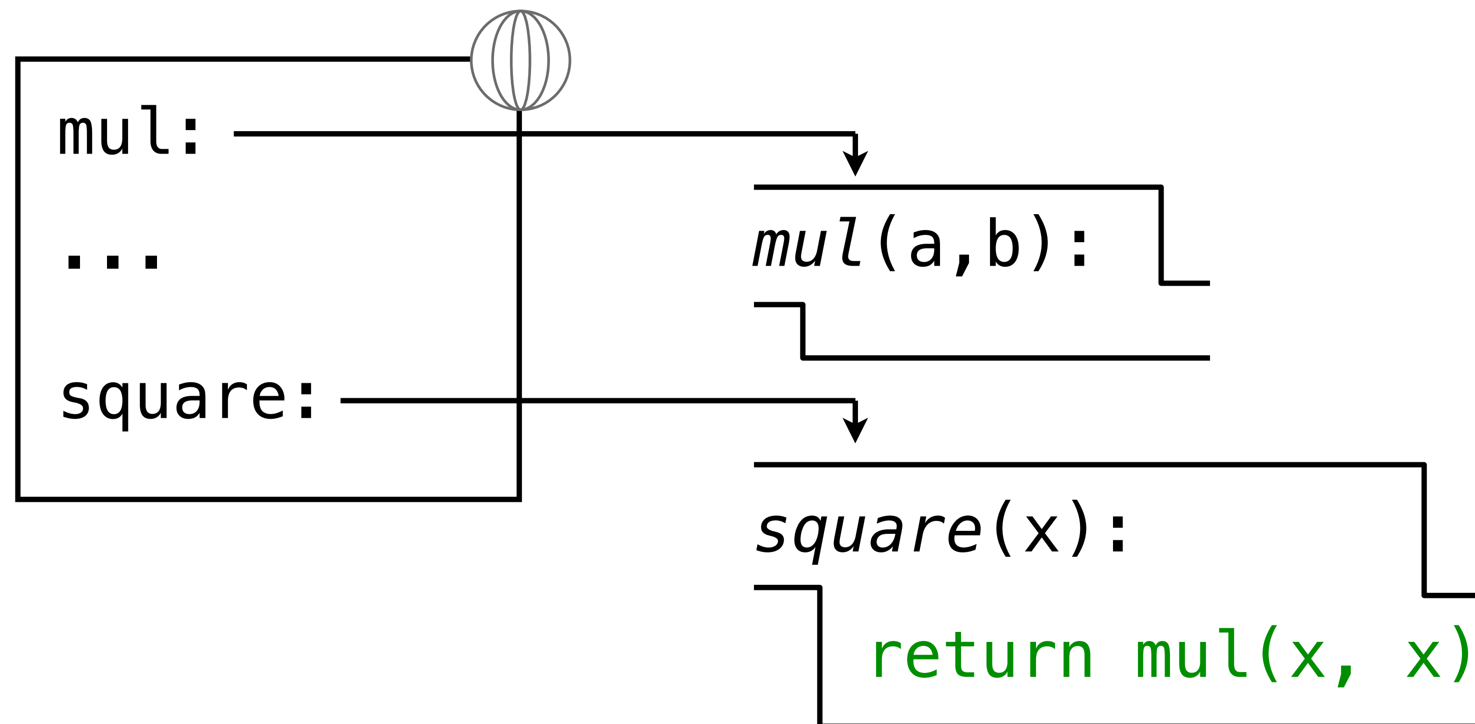
Expressions (Program):

Expressions are Python code

Not part of an environment

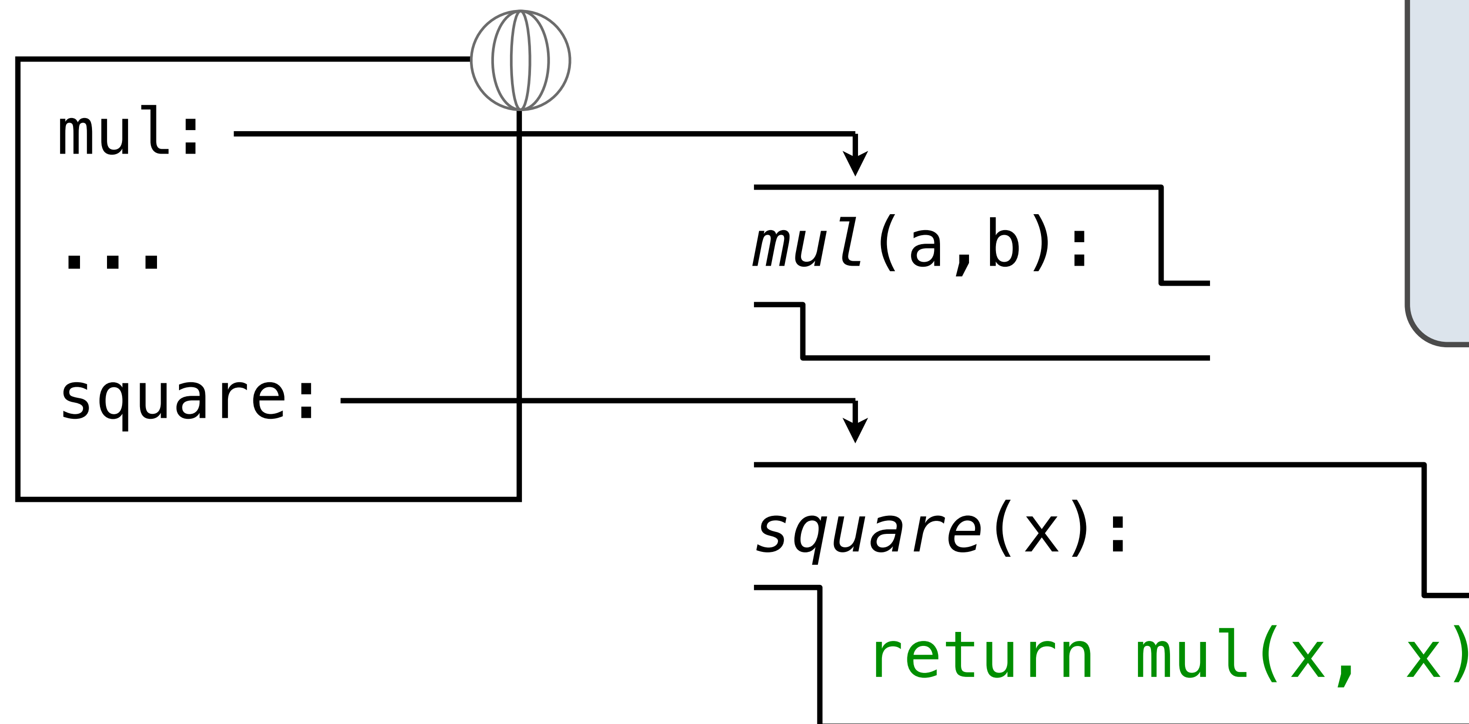
They are evaluated in an environment to yield a value

Multiple Environments in One Diagram!




```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

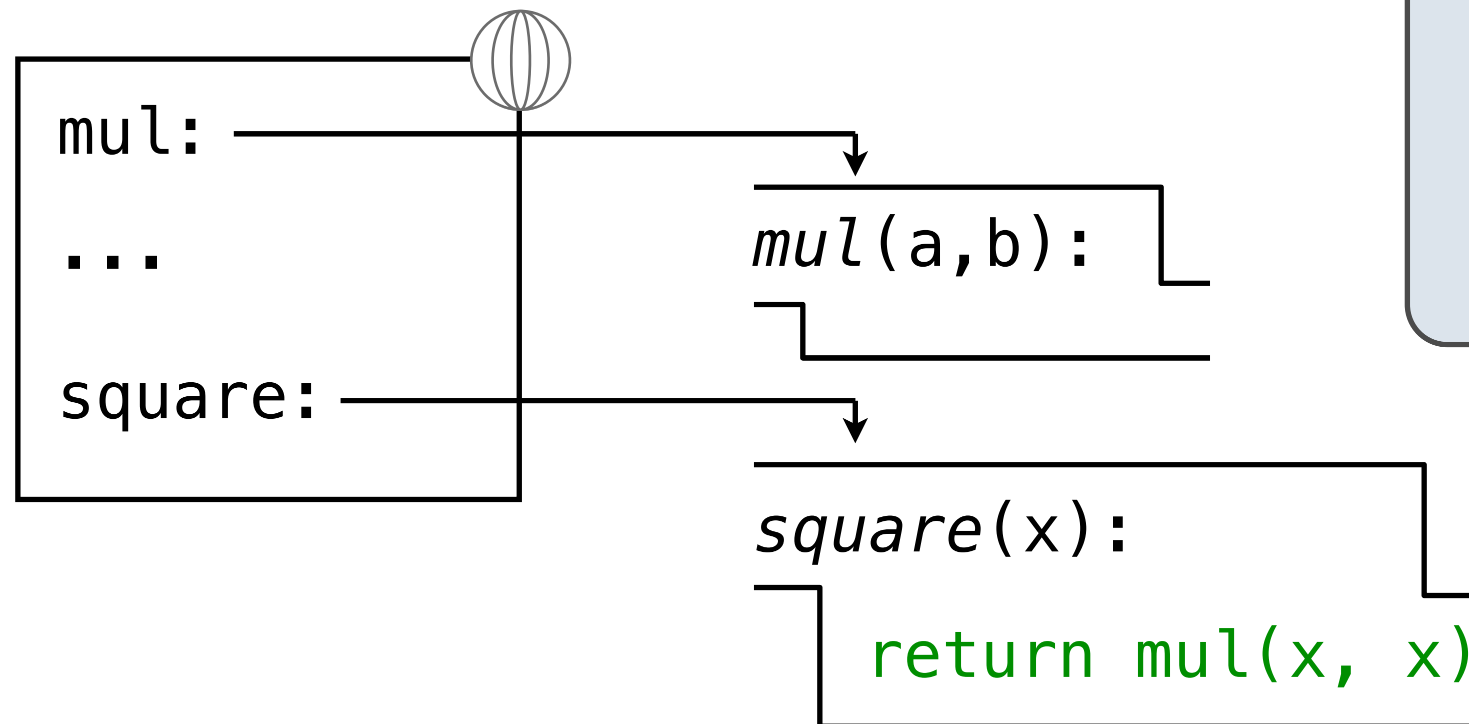
Multiple Environments in One Diagram!



Every call to a user-defined function creates a new local frame

```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

Multiple Environments in One Diagram!

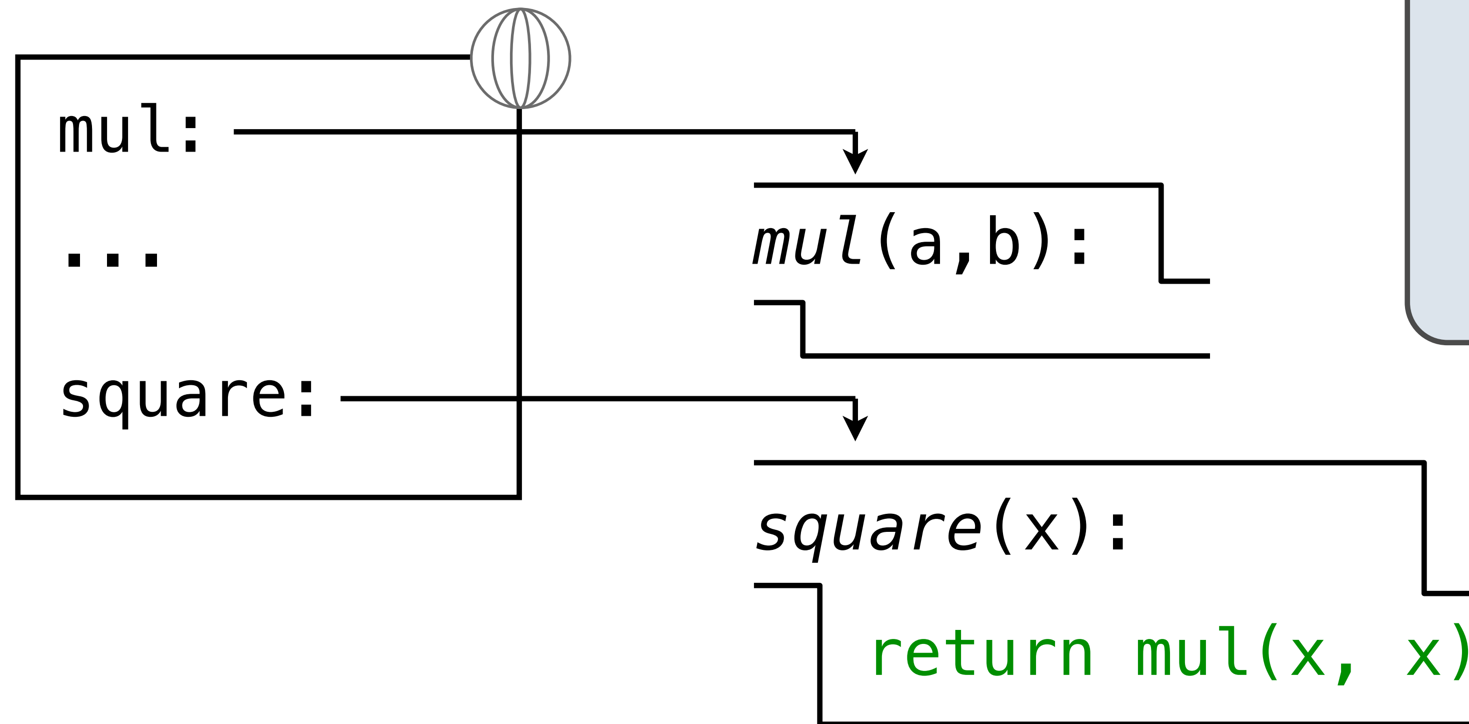


Every call to a user-defined function creates a new local frame

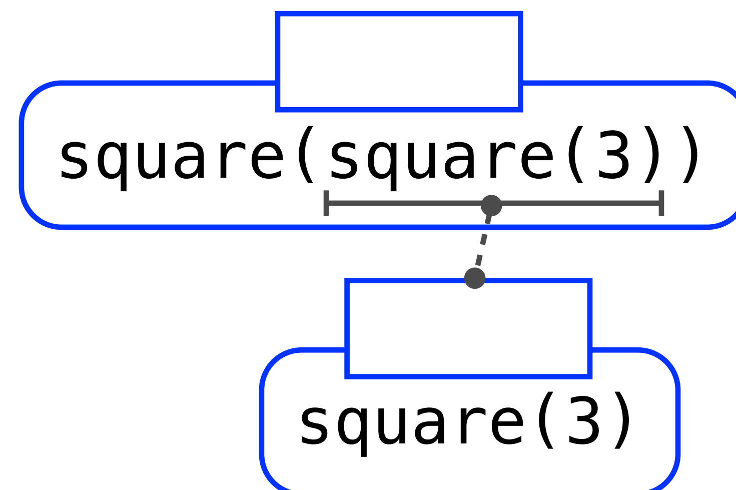
square(square(3))

```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

Multiple Environments in One Diagram!

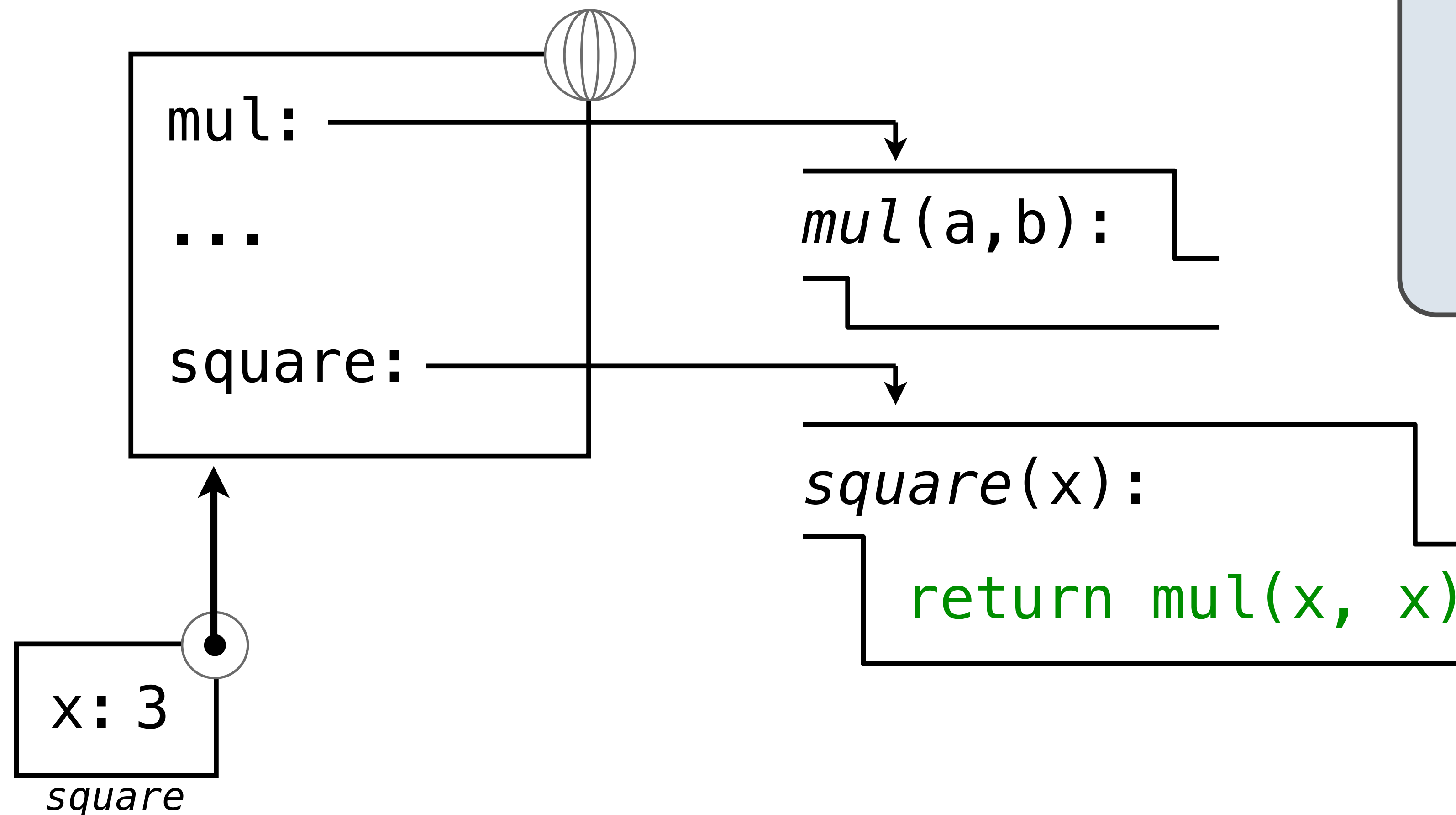


Every call to a user-defined function creates a new local frame

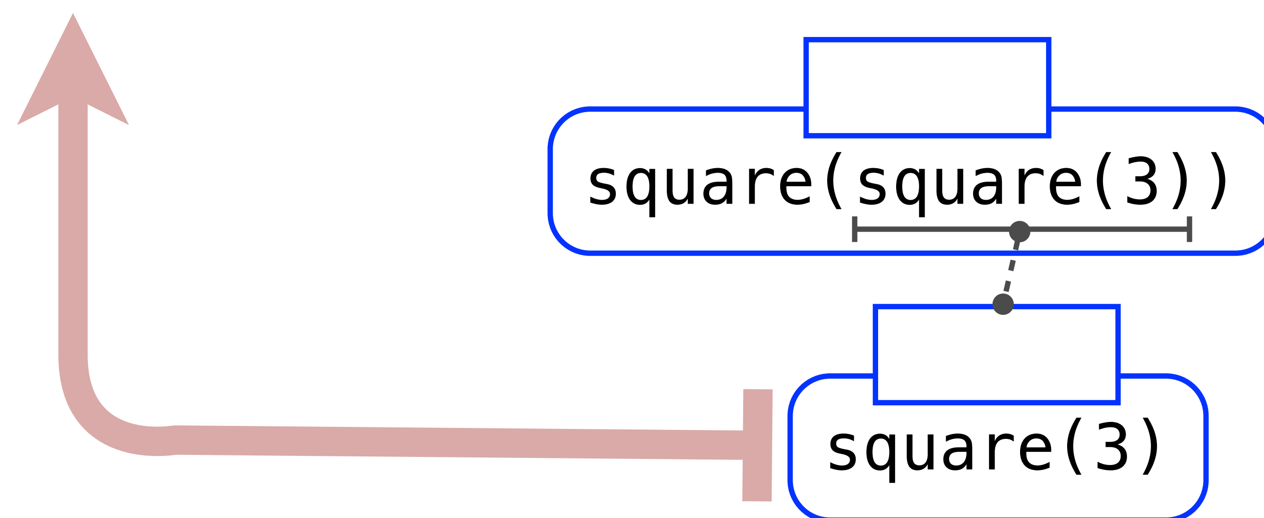



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

Multiple Environments in One Diagram!

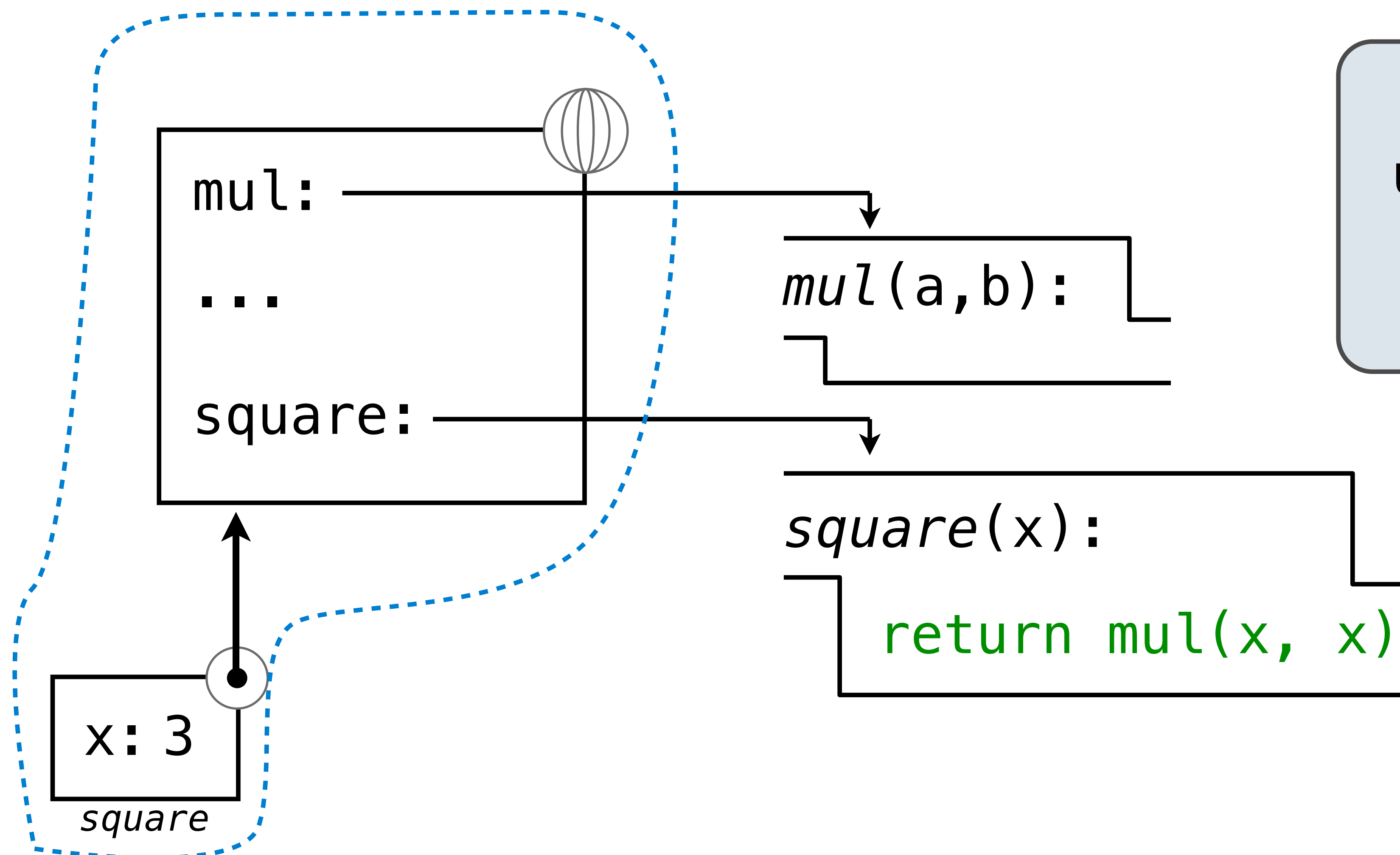


Every call to a user-defined function creates a new local frame

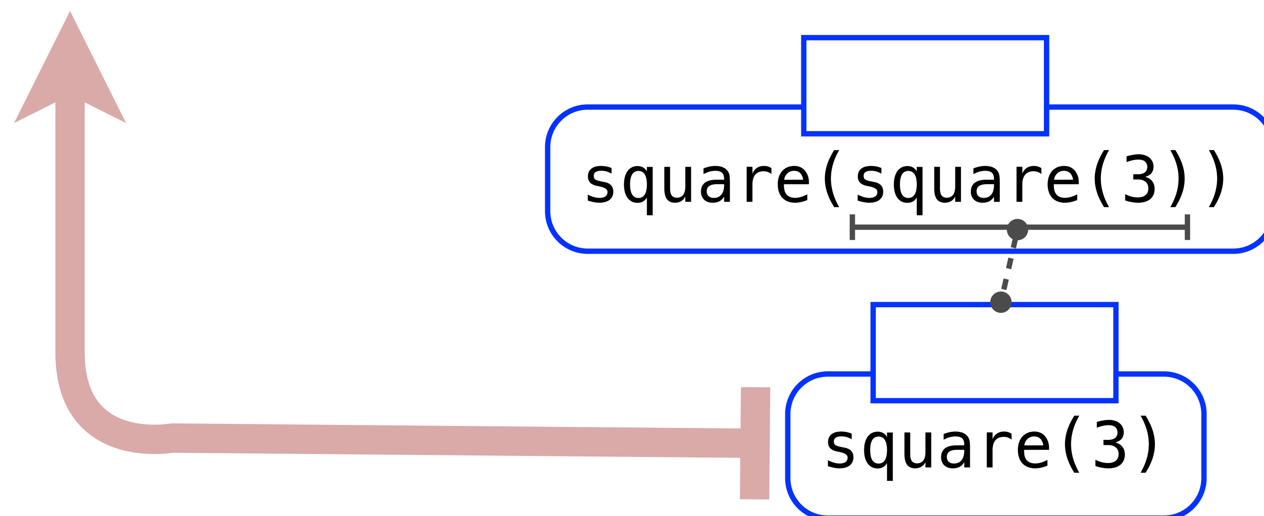



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

Multiple Environments in One Diagram!

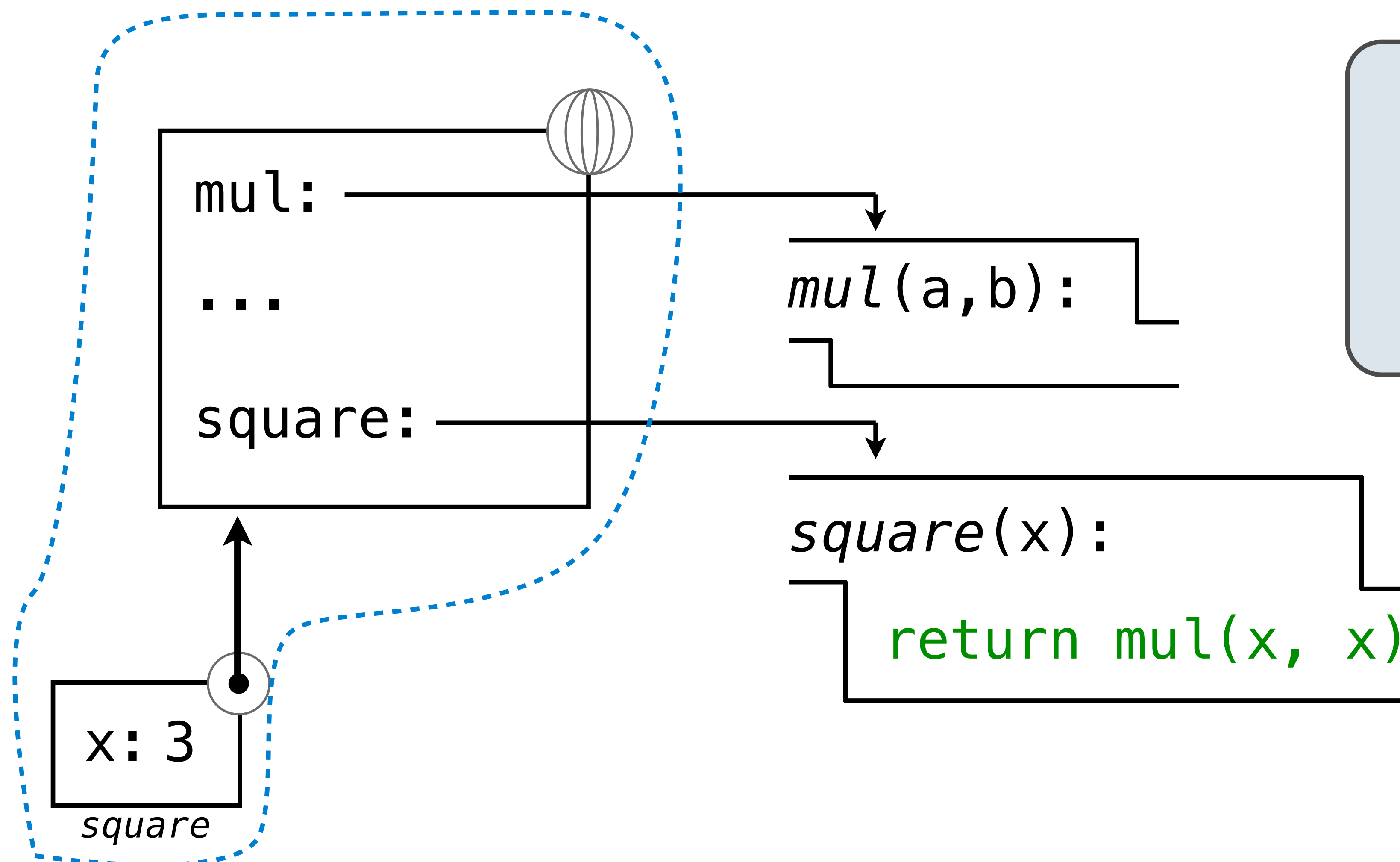


Every call to a user-defined function creates a new local frame

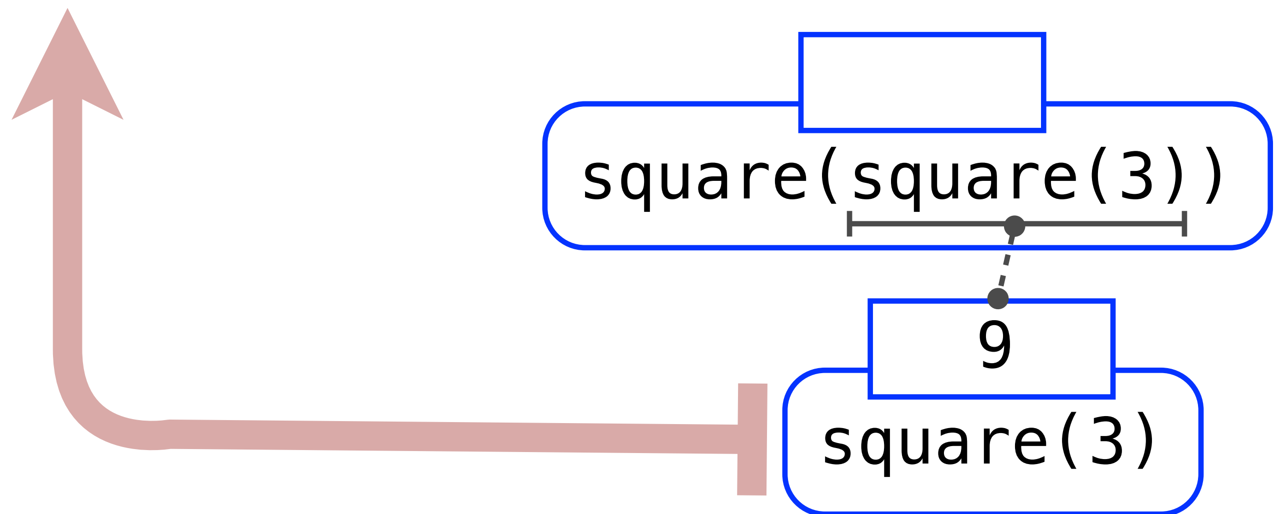



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

Multiple Environments in One Diagram!

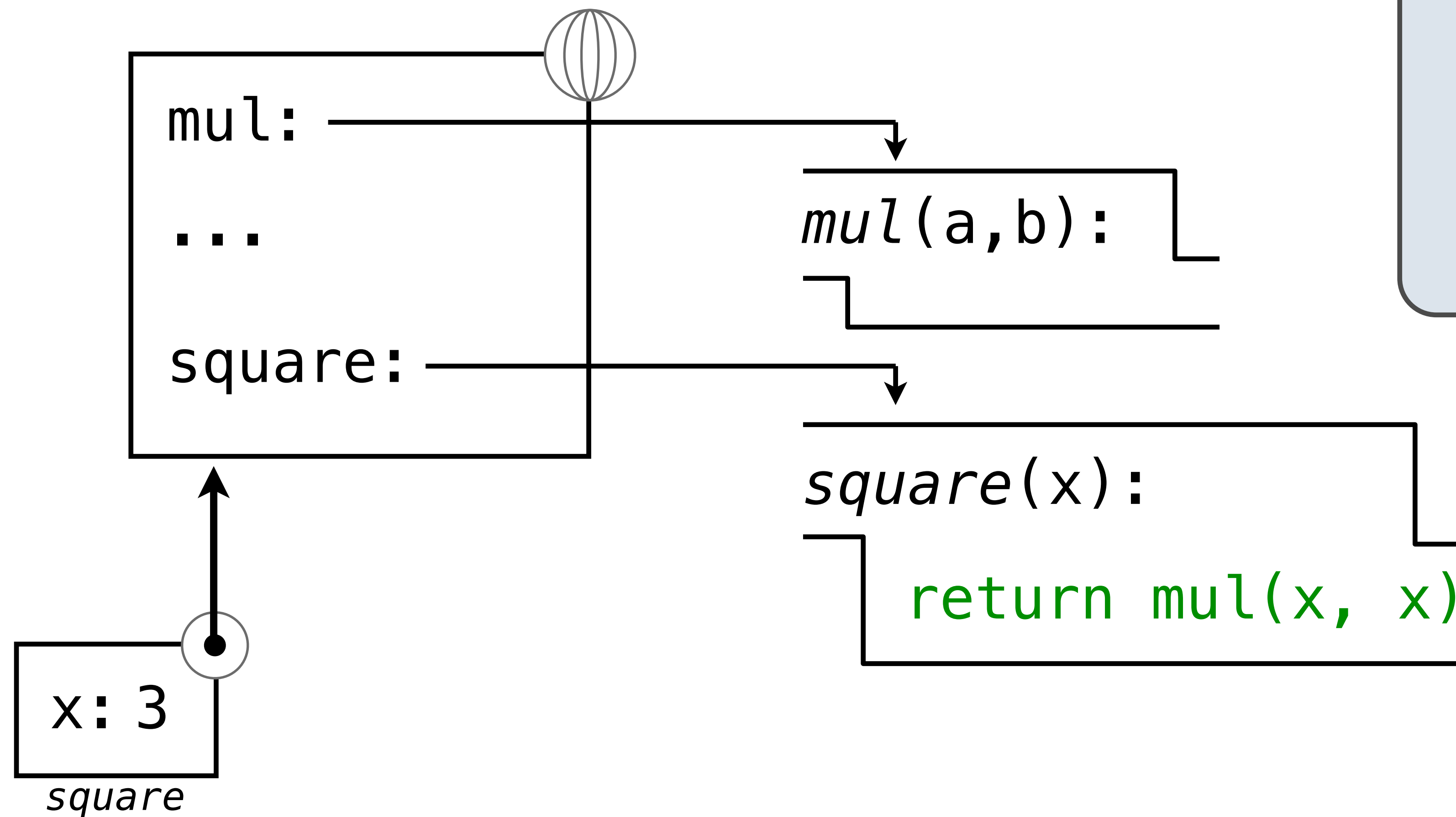


Every call to a user-defined function creates a new local frame

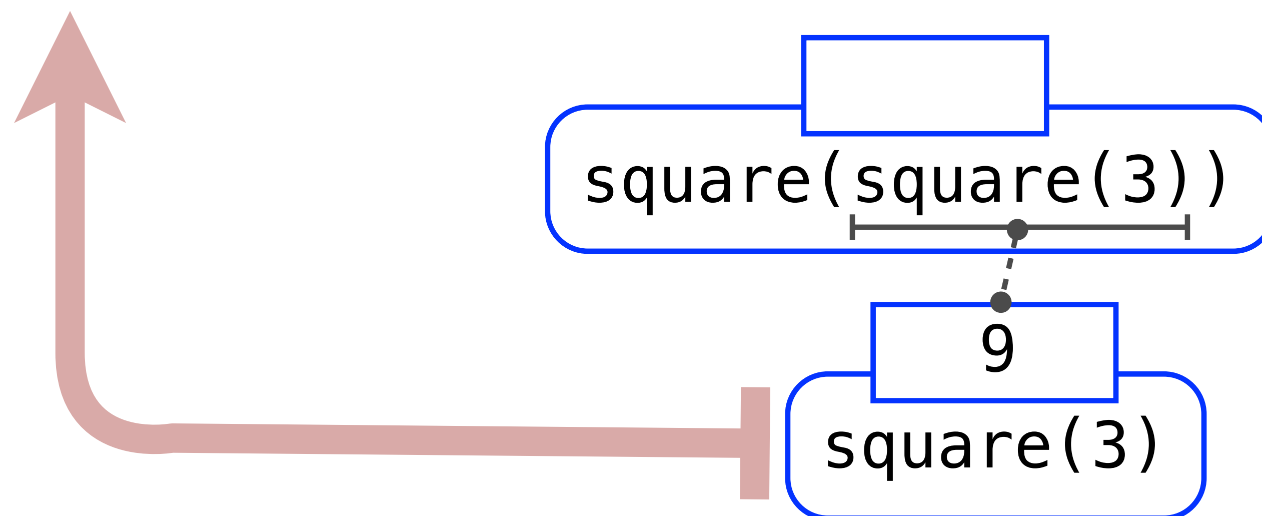



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

Multiple Environments in One Diagram!

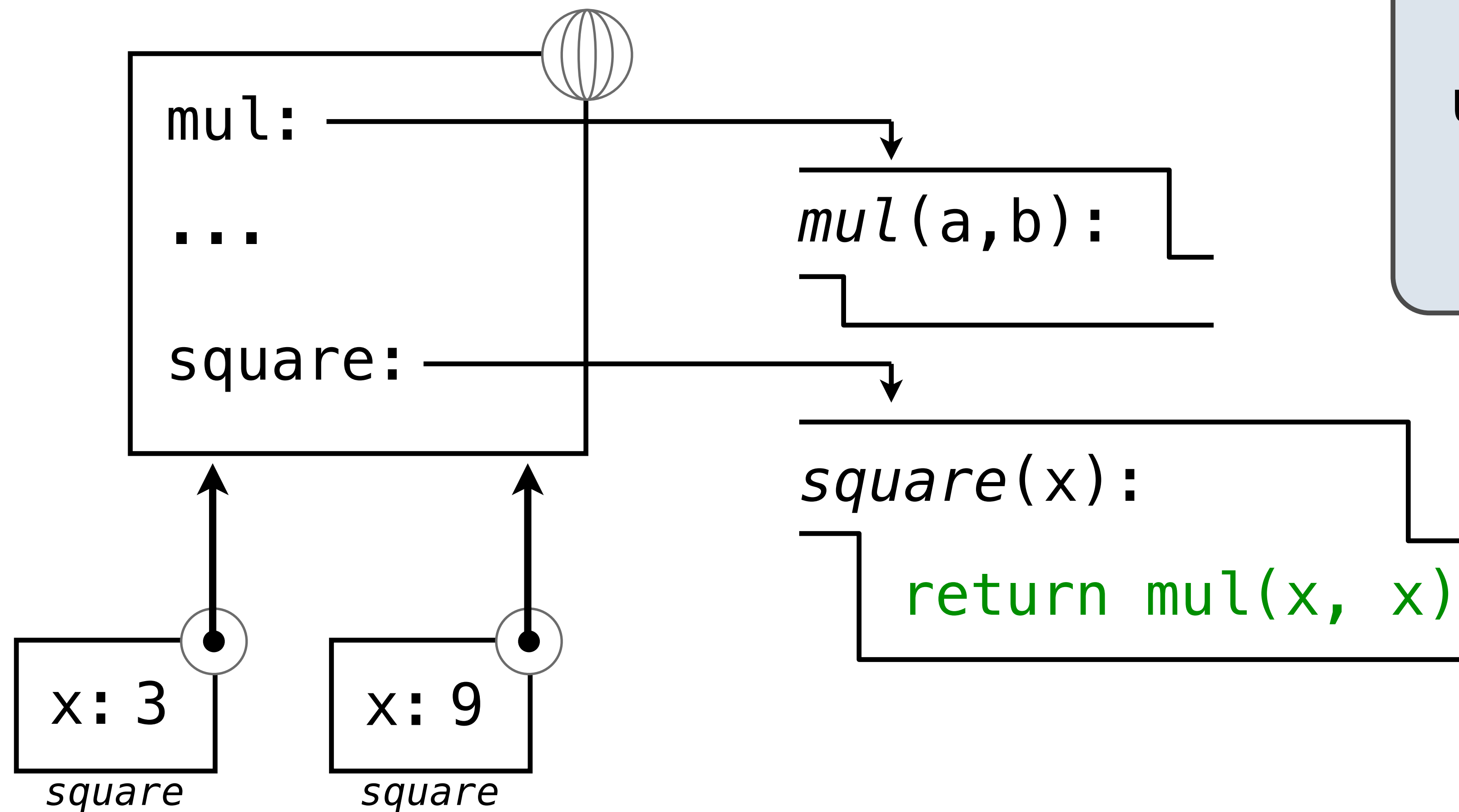


Every call to a user-defined function creates a new local frame

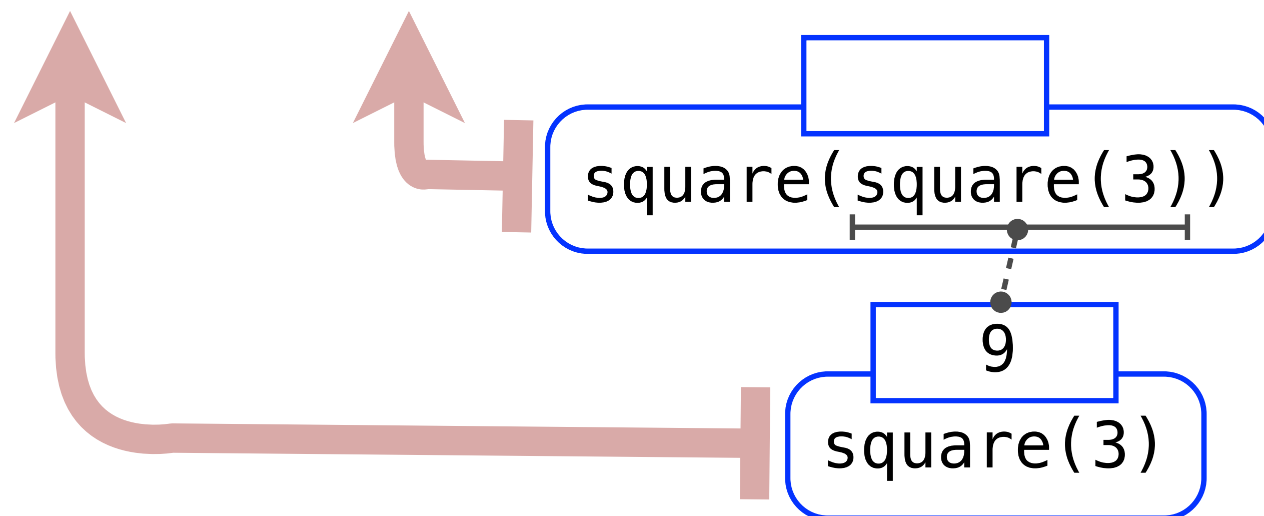



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

Multiple Environments in One Diagram!

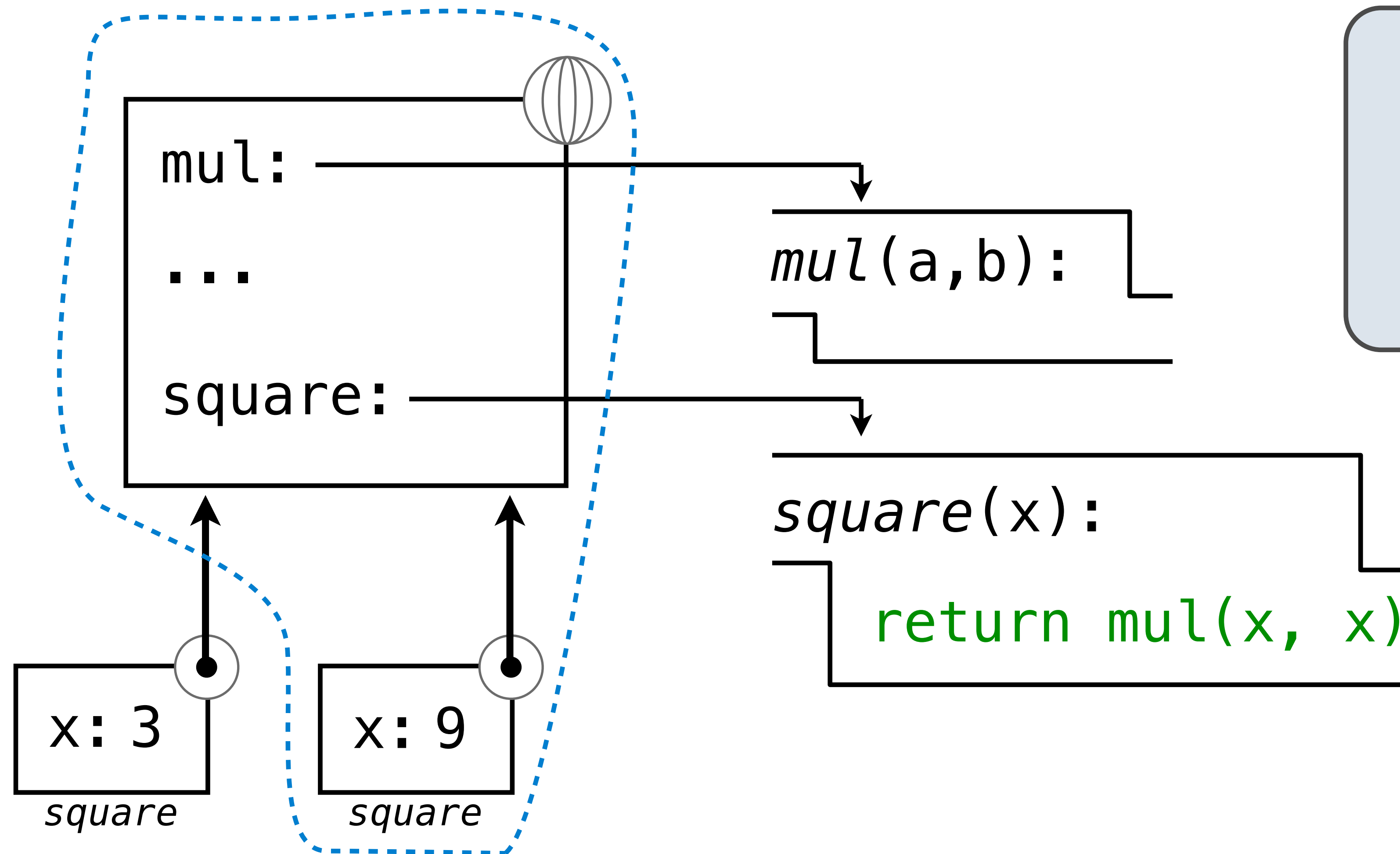


Every call to a user-defined function creates a new local frame

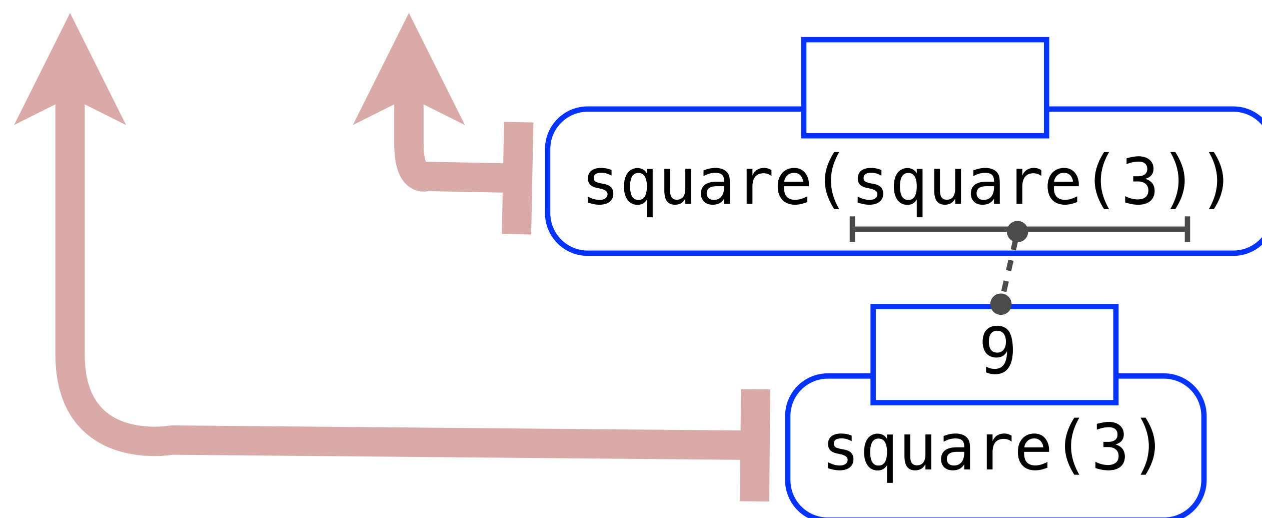



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

Multiple Environments in One Diagram!

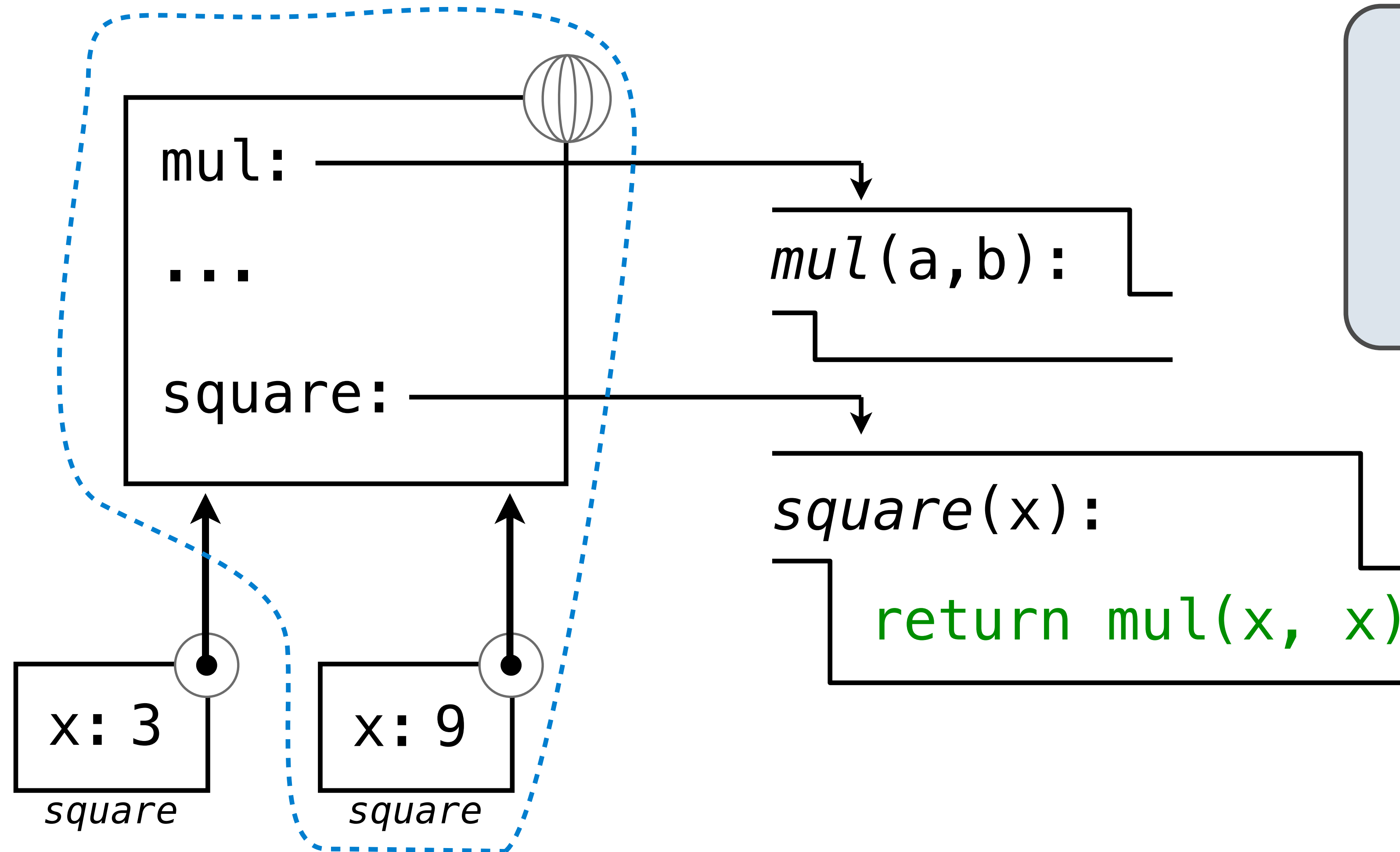


Every call to a user-defined function creates a new local frame

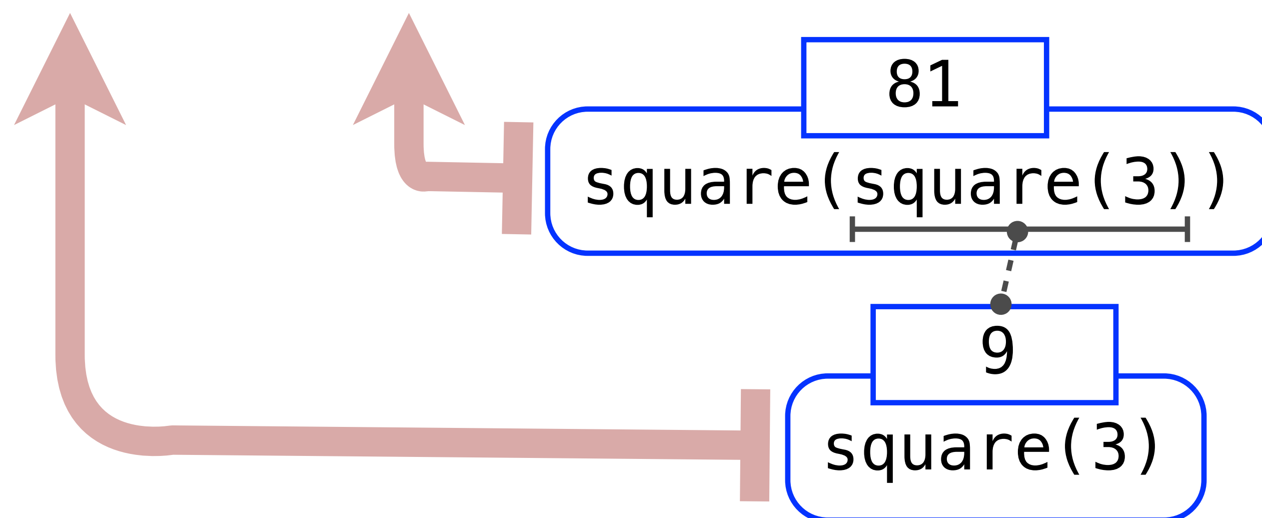



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

Multiple Environments in One Diagram!

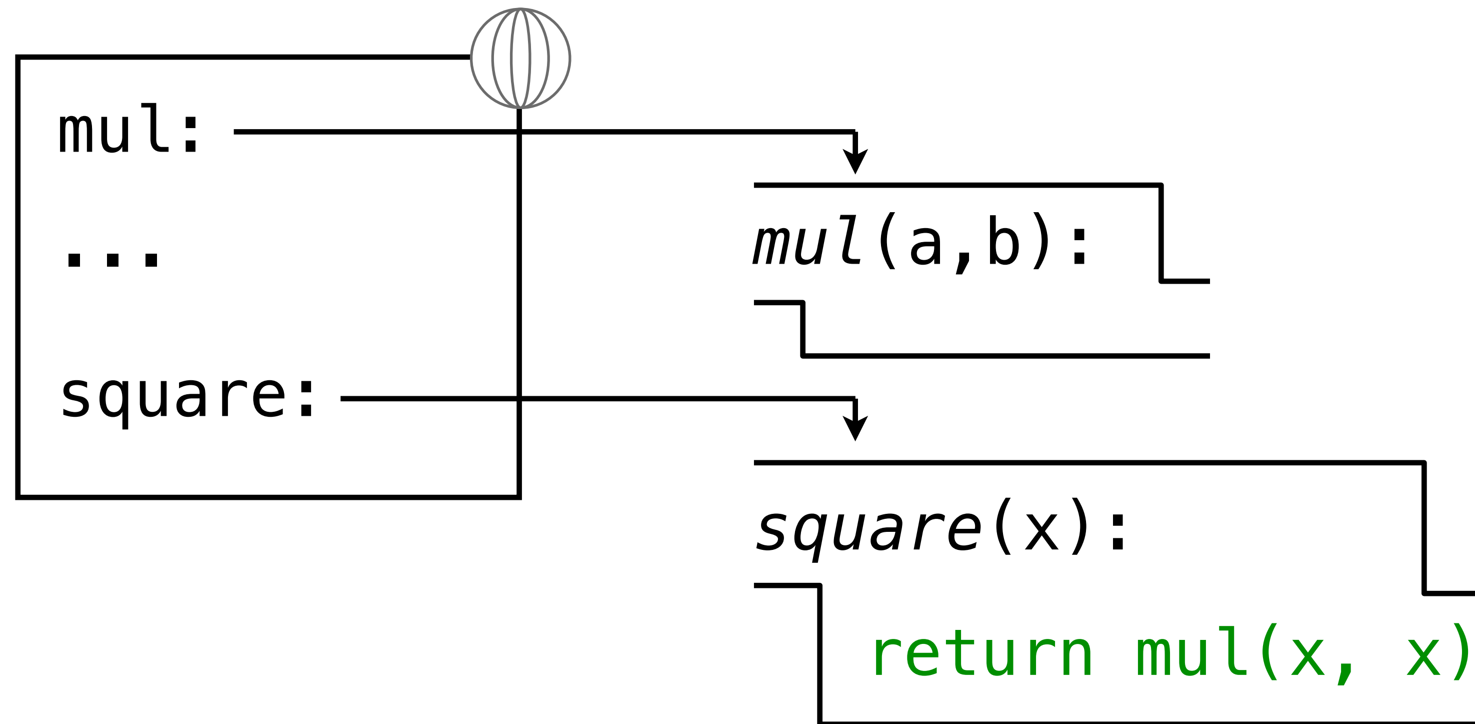


Every call to a user-defined function creates a new local frame



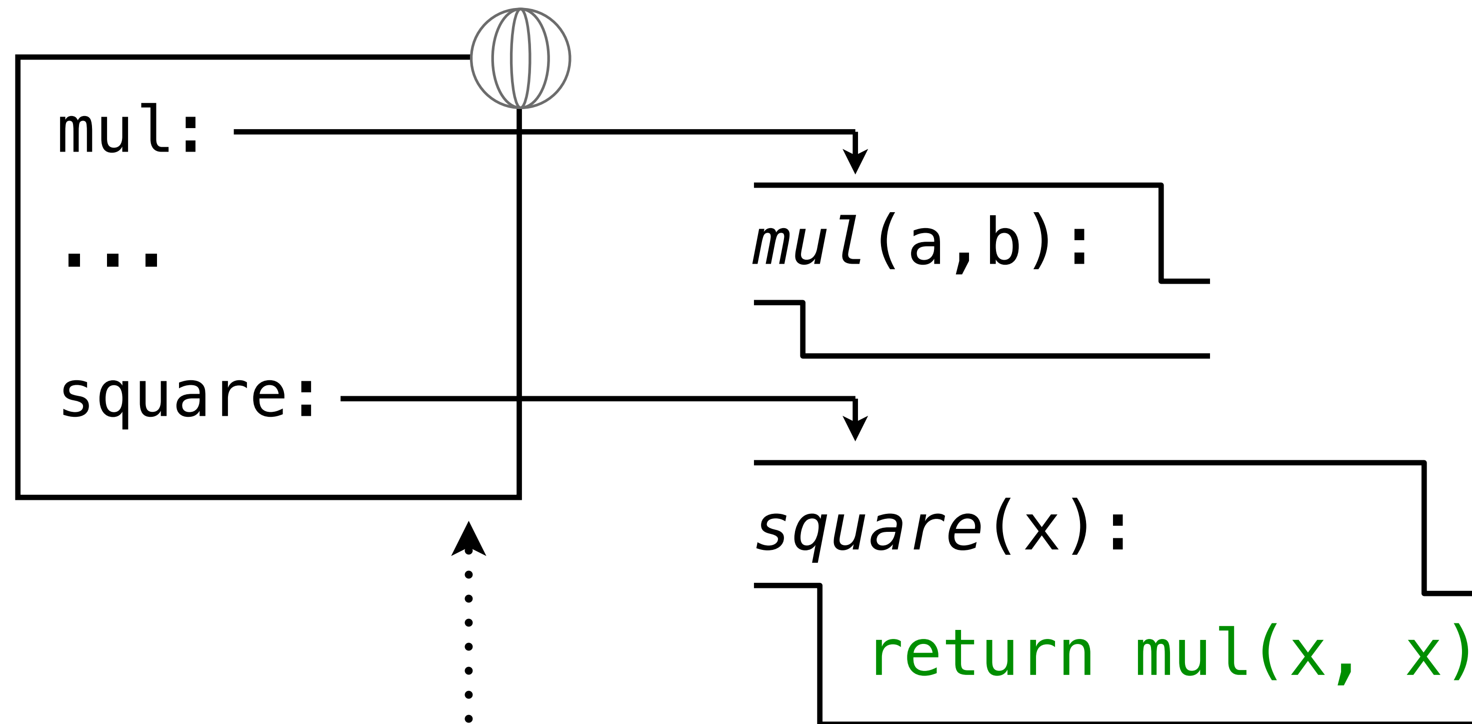

```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

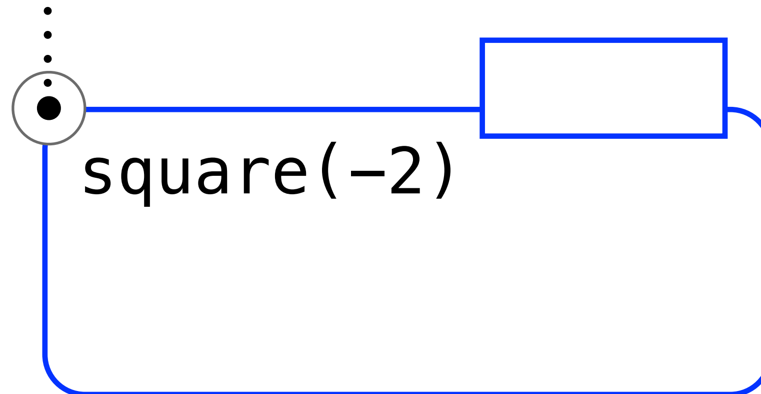
Names Have No Meaning Without Environments




```
from operator import mul
def square(x):
    return mul(x, x)
square(-2)
```

Names Have No Meaning Without Environments

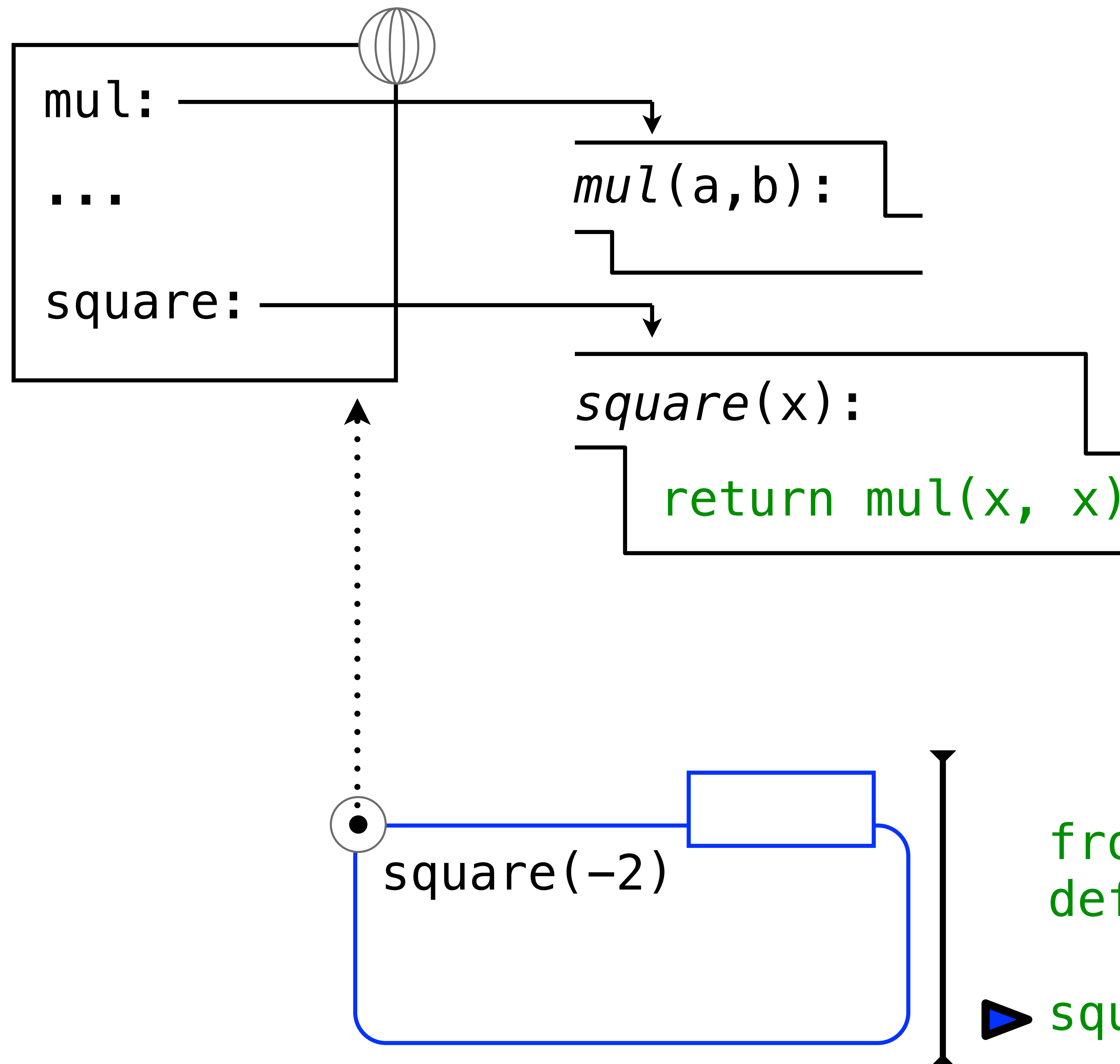





```
from operator import mul
def square(x):
    return mul(x, x)
square(-2)
```

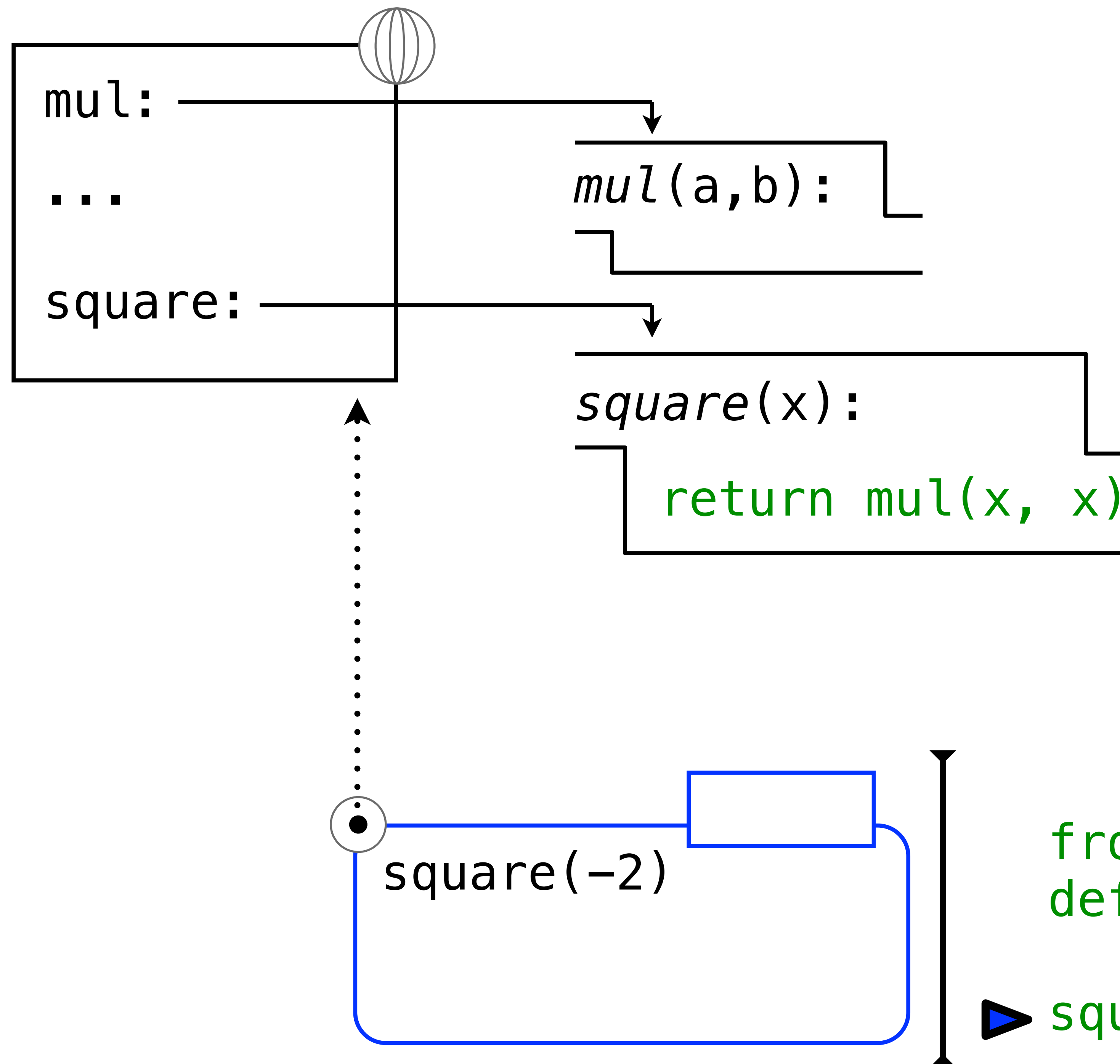
Names Have No Meaning Without Environments

A name evaluates to the **value bound** to that name




```
from operator import mul
def square(x):
    return mul(x, x)
square(-2)
```

Names Have No Meaning Without Environments

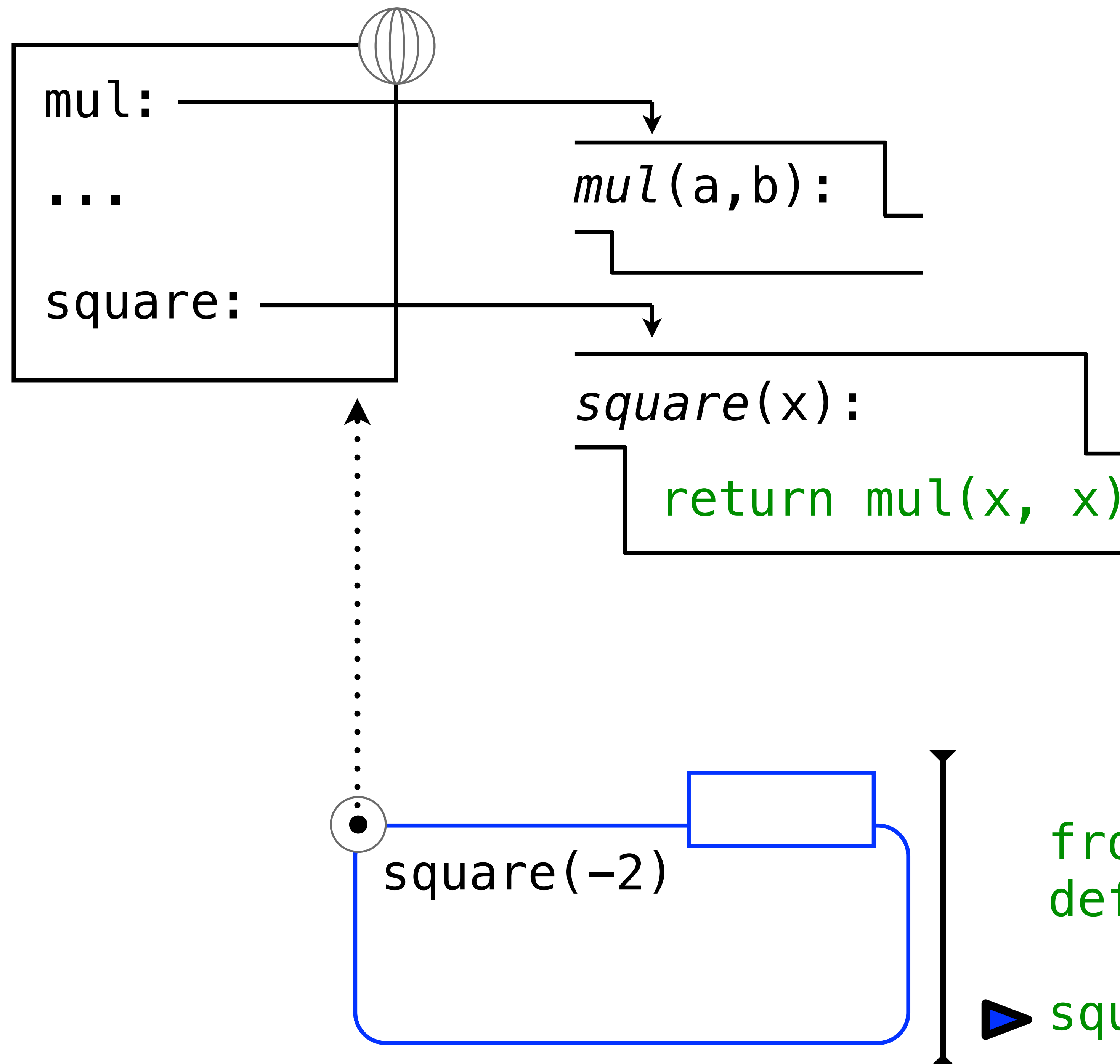


A name evaluates to the **value** **bound** to that name

...in the **earliest** frame of the **current** **environment**

```
from operator import mul
def square(x):
    return mul(x, x)
square(-2)
```

Names Have No Meaning Without Environments



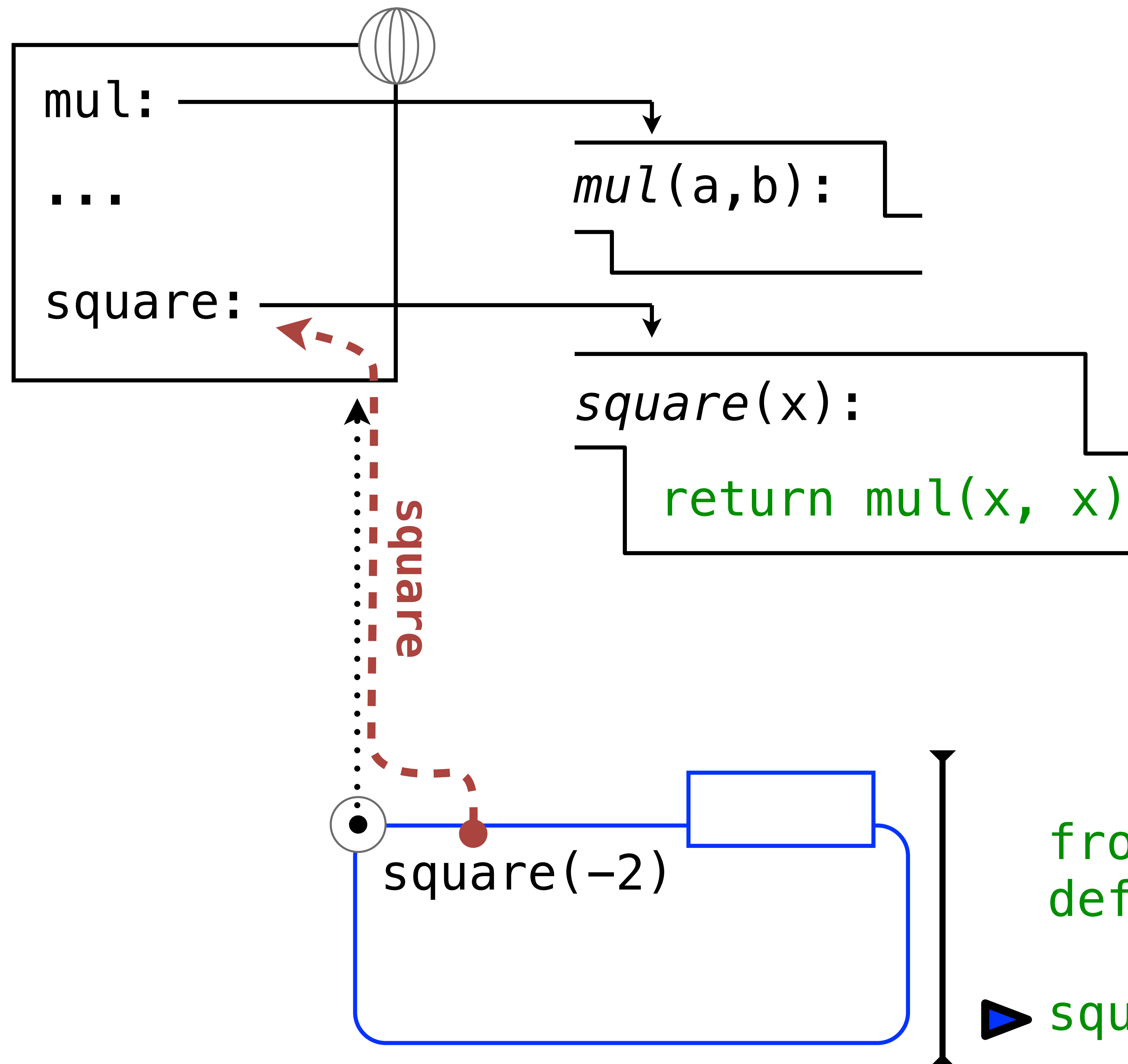
A name evaluates to the **value bound** to that name

...in the **earliest** frame of the **current environment**

...in which that **name is found**

```
from operator import mul
def square(x):
    return mul(x, x)
square(-2)
```

Names Have No Meaning Without Environments



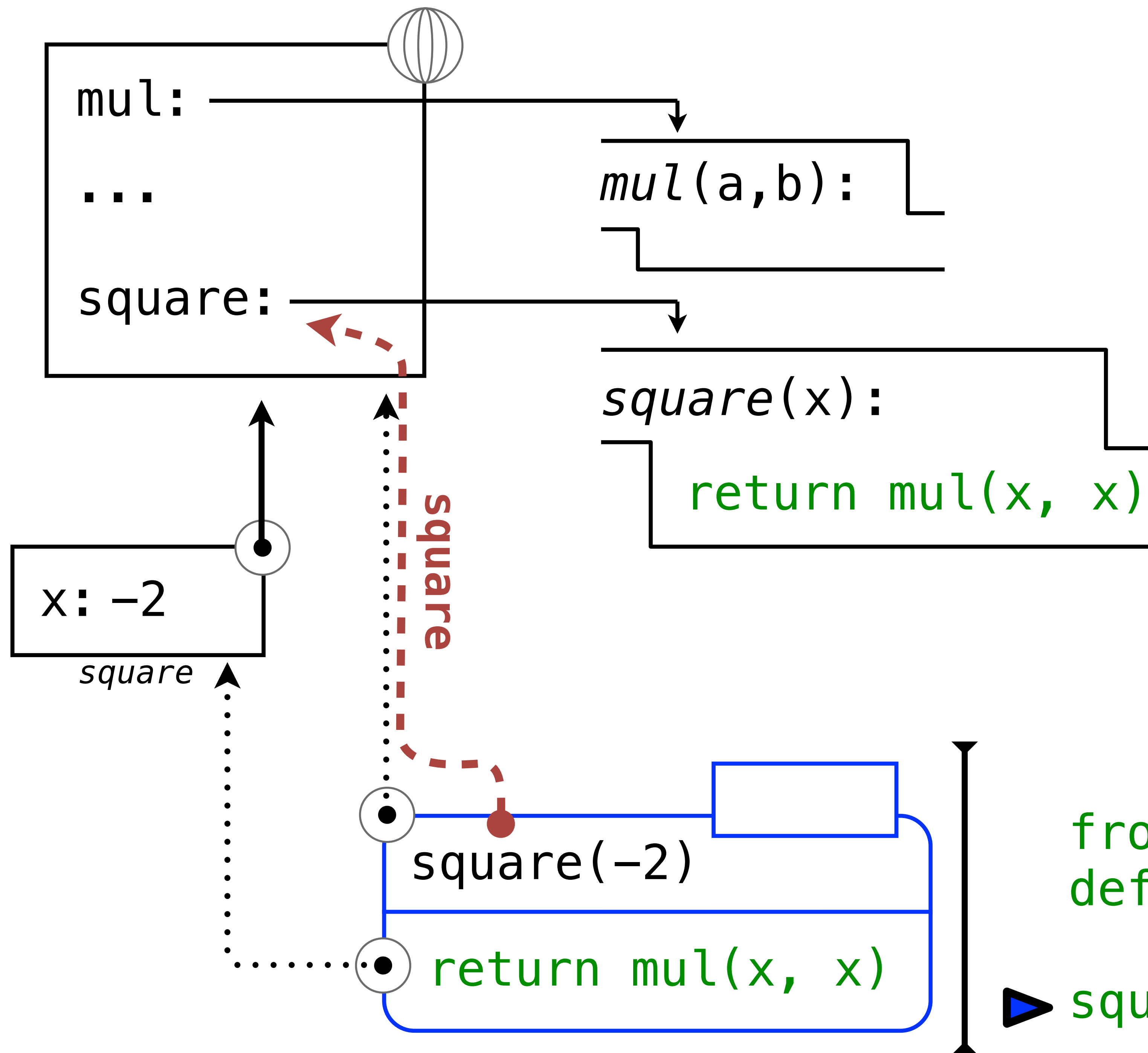
A name evaluates to the **value bound** to that name

...in the **earliest** frame of the **current environment**

...in which that **name is found**

```
from operator import mul
def square(x):
    return mul(x, x)
square(-2)
```

Names Have No Meaning Without Environments



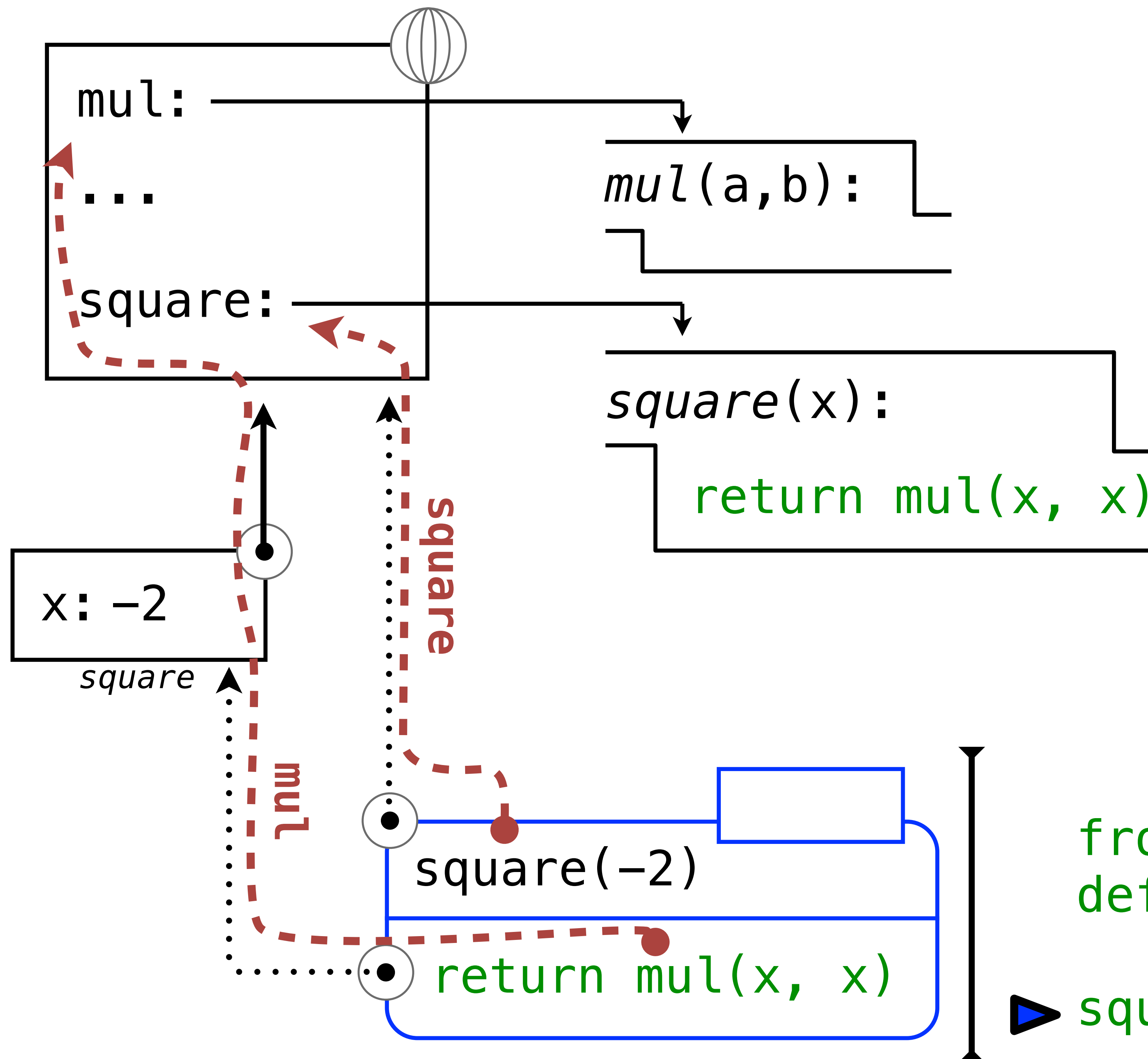
A name evaluates to the **value bound** to that name

...in the **earliest** frame of the **current environment**

...in which that **name is found**

```
from operator import mul
def square(x):
    return mul(x, x)
square(-2)
```

Names Have No Meaning Without Environments



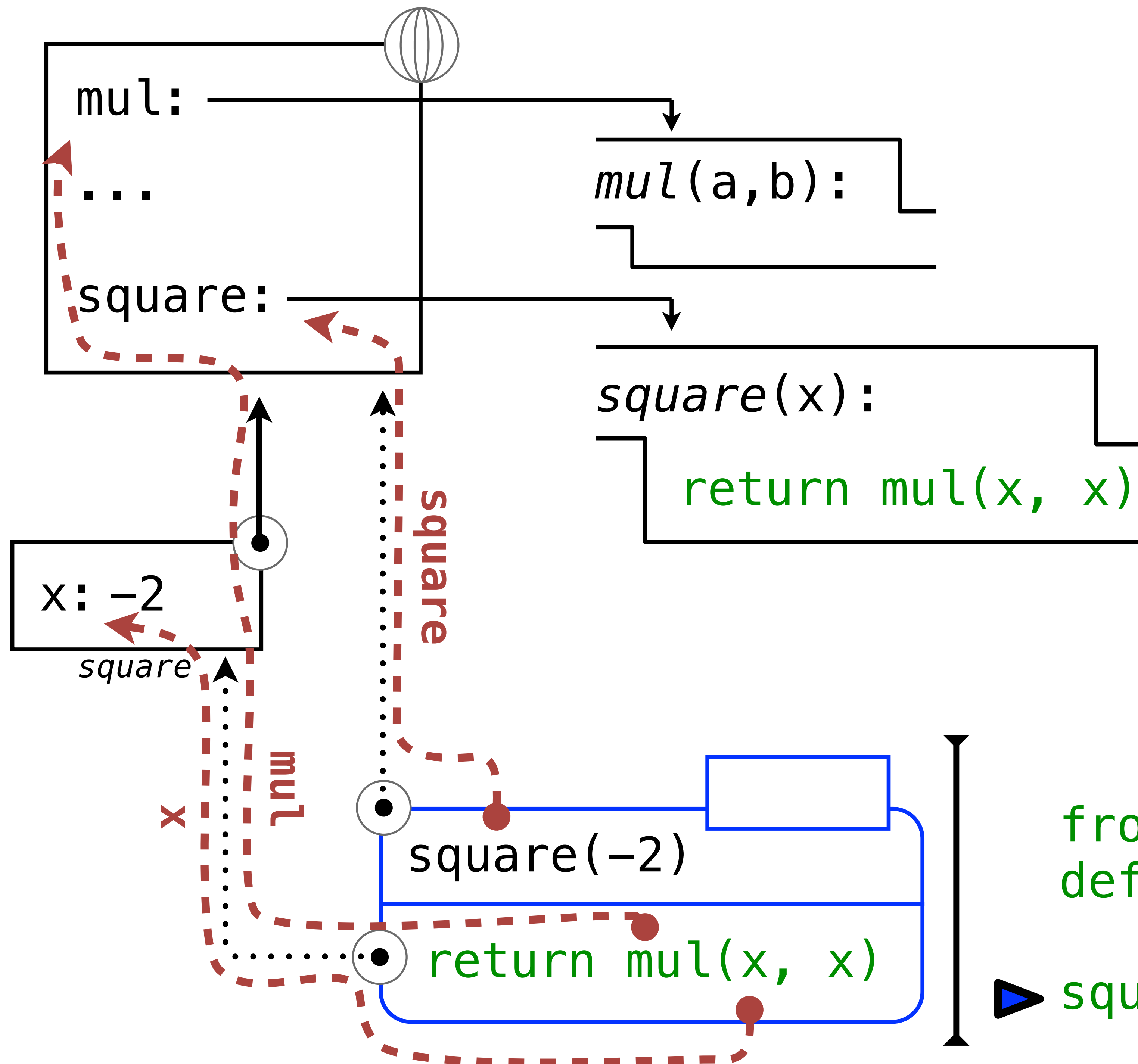
A name evaluates to the **value bound** to that name

...in the **earliest** frame of the **current environment**

...in which that **name is found**

```
from operator import mul
def square(x):
    return mul(x, x)
square(-2)
```

Names Have No Meaning Without Environments



A name evaluates to the **value bound** to that name

...in the **earliest** frame of the **current environment**

...in which that **name is found**

```
from operator import mul
def square(x):
    return mul(x, x)
square(-2)
```

Formal Parameters

Formal Parameters

```
def square(x):
    return mul(x, x)
```

Formal Parameters

```
def square(x):
    return mul(x, x)
```

vs

Formal Parameters

```
def square(x):                      def square(y):
    return mul(x, x)        vs          return mul(y, y)
```

Formal Parameters

```
def square(x):
    return mul(x, x)
```

vs

```
def square(y):
    return mul(y, y)
```

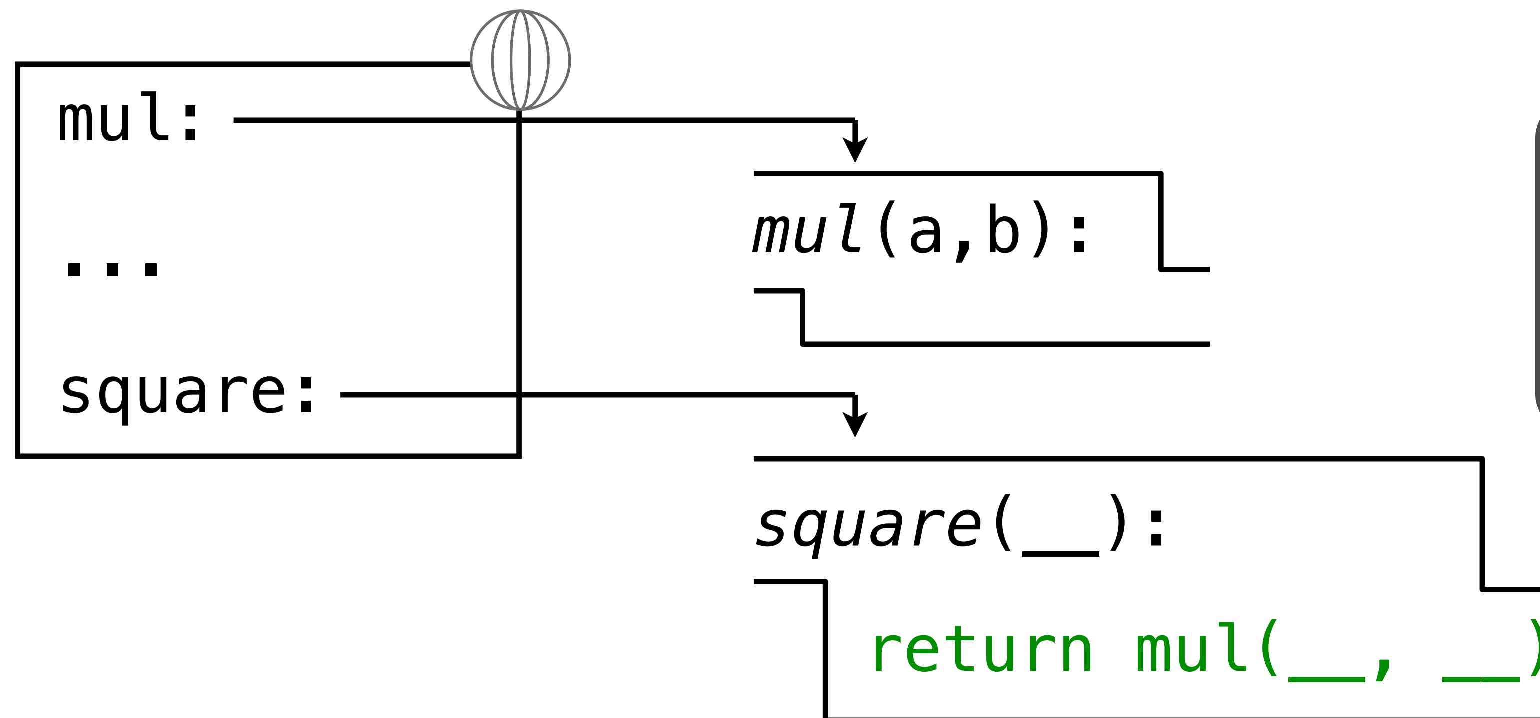
Formal parameter names stay *local* to their frame

Formal Parameters

```
def square(x):
    return mul(x, x)
```

vs

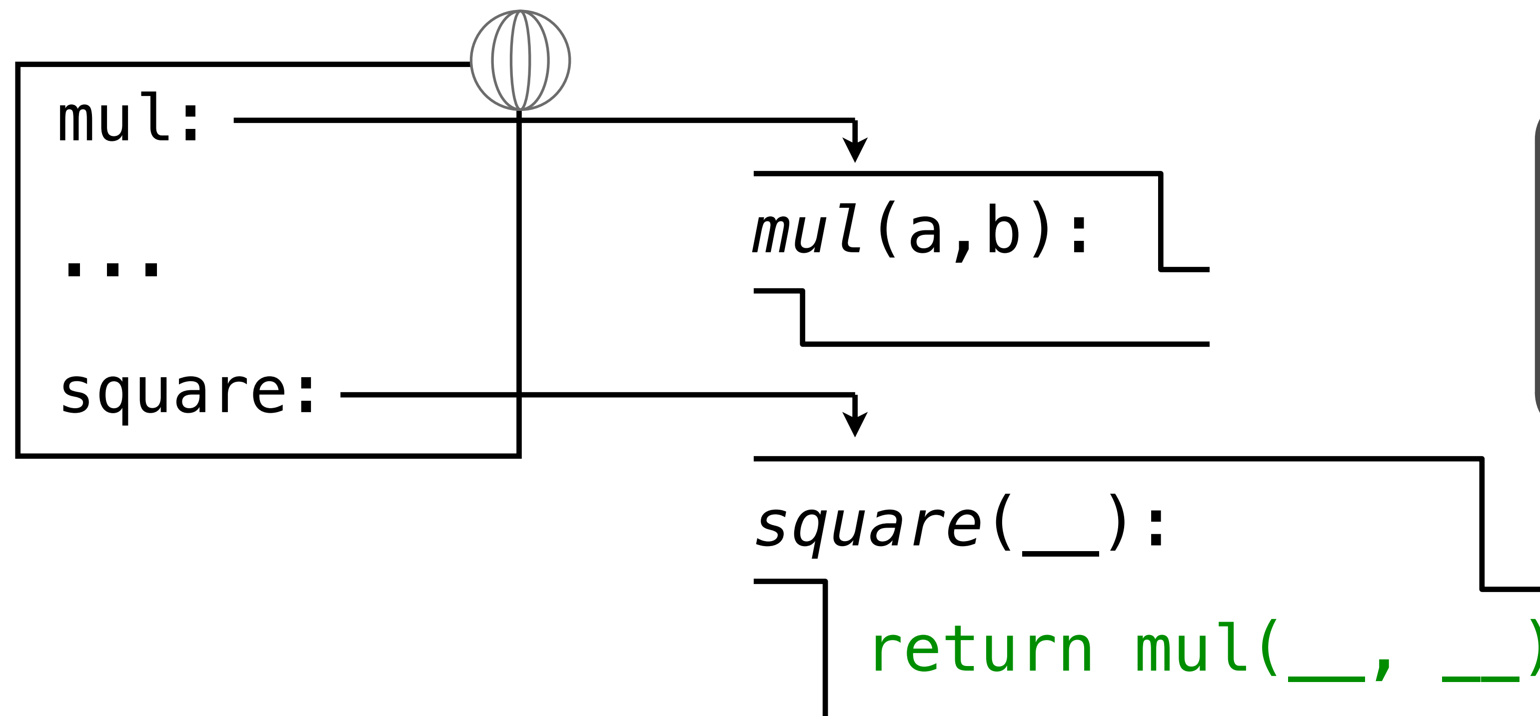
```
def square(y):
    return mul(y, y)
```



Formal parameter names stay *local* to their frame

Formal Parameters

```
def square(x):                    def square(y):
    return mul(x, x)      vs          return mul(y, y)
```



Formal parameter names stay *local* to their frame

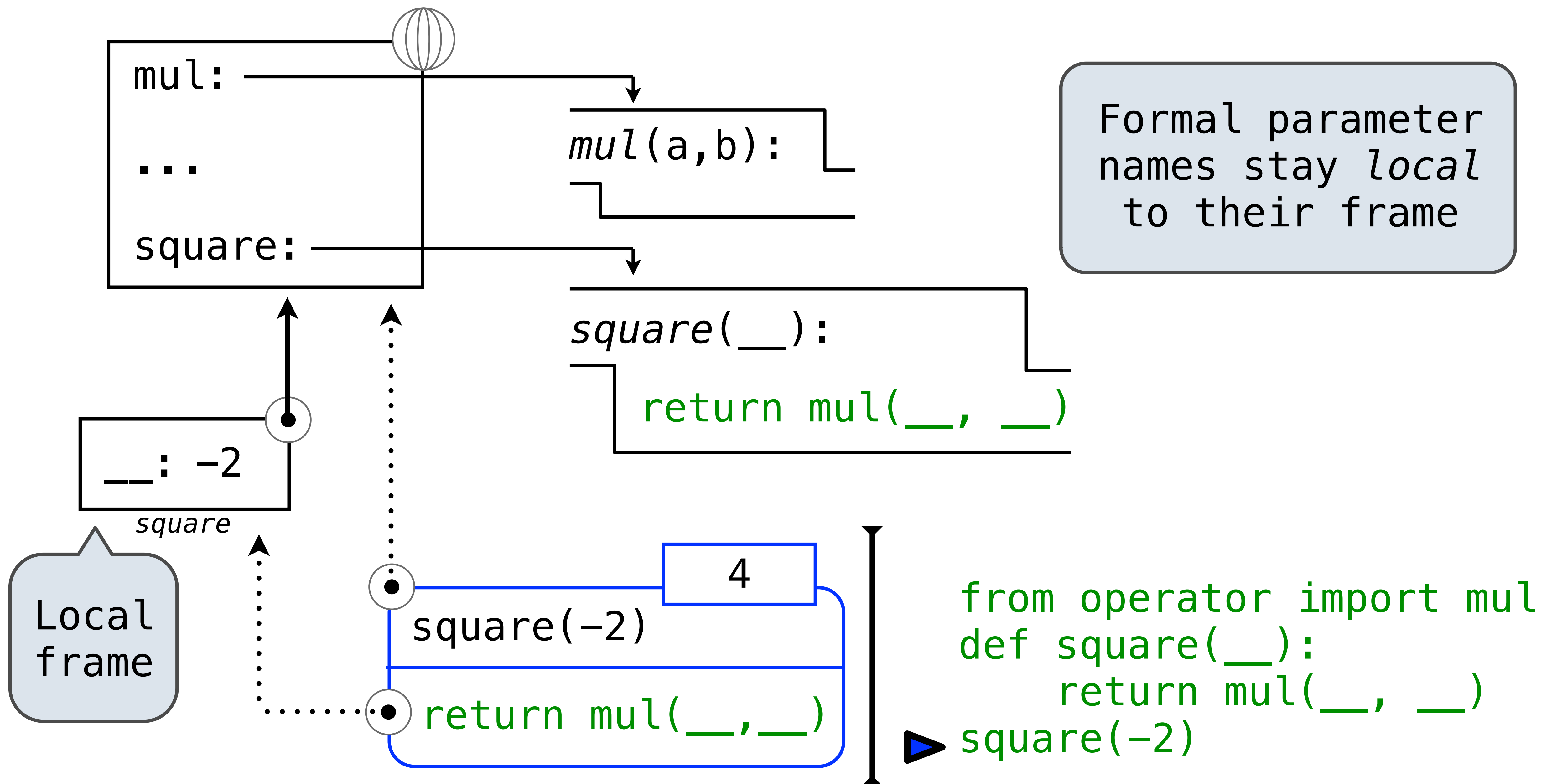
```
from operator import mul
def square(__):
    return mul(__, __)
square(-2)
```

Formal Parameters

```
def square(x):
    return mul(x, x)
```

vs

```
def square(y):
    return mul(y, y)
```



Shadowing Names

Careful!

```
def square(mul):
    return mul(mul, mul)
square(-2)
```

Shadowing Names

Careful!

```
def square(mul):
    return mul(mul, mul)
square(-2)
```

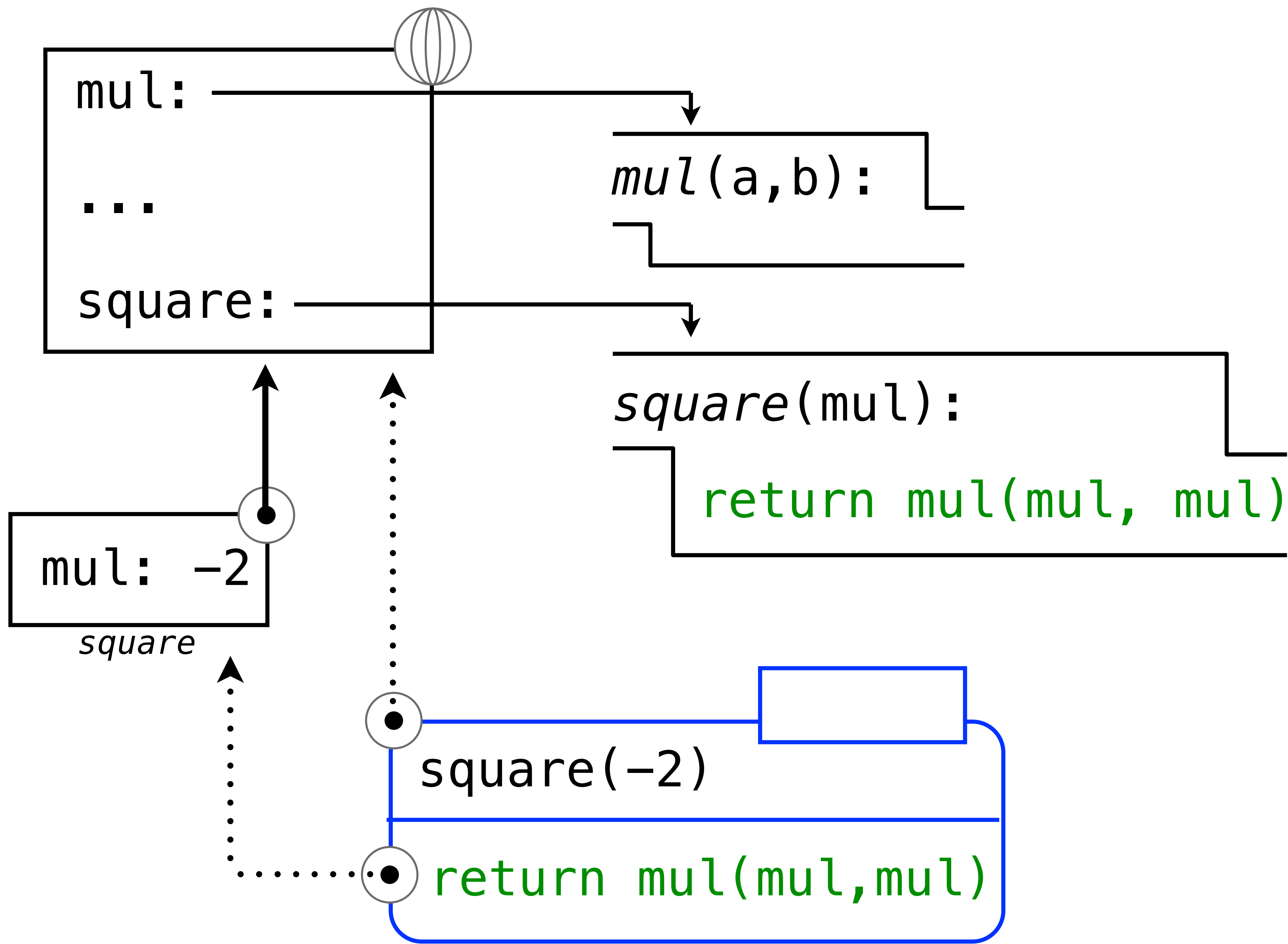
If we use this formal parameter

Shadowing Names

Careful!

```
def square(mul):
    return mul(mul, mul)
square(-2)
```

If we use this formal parameter

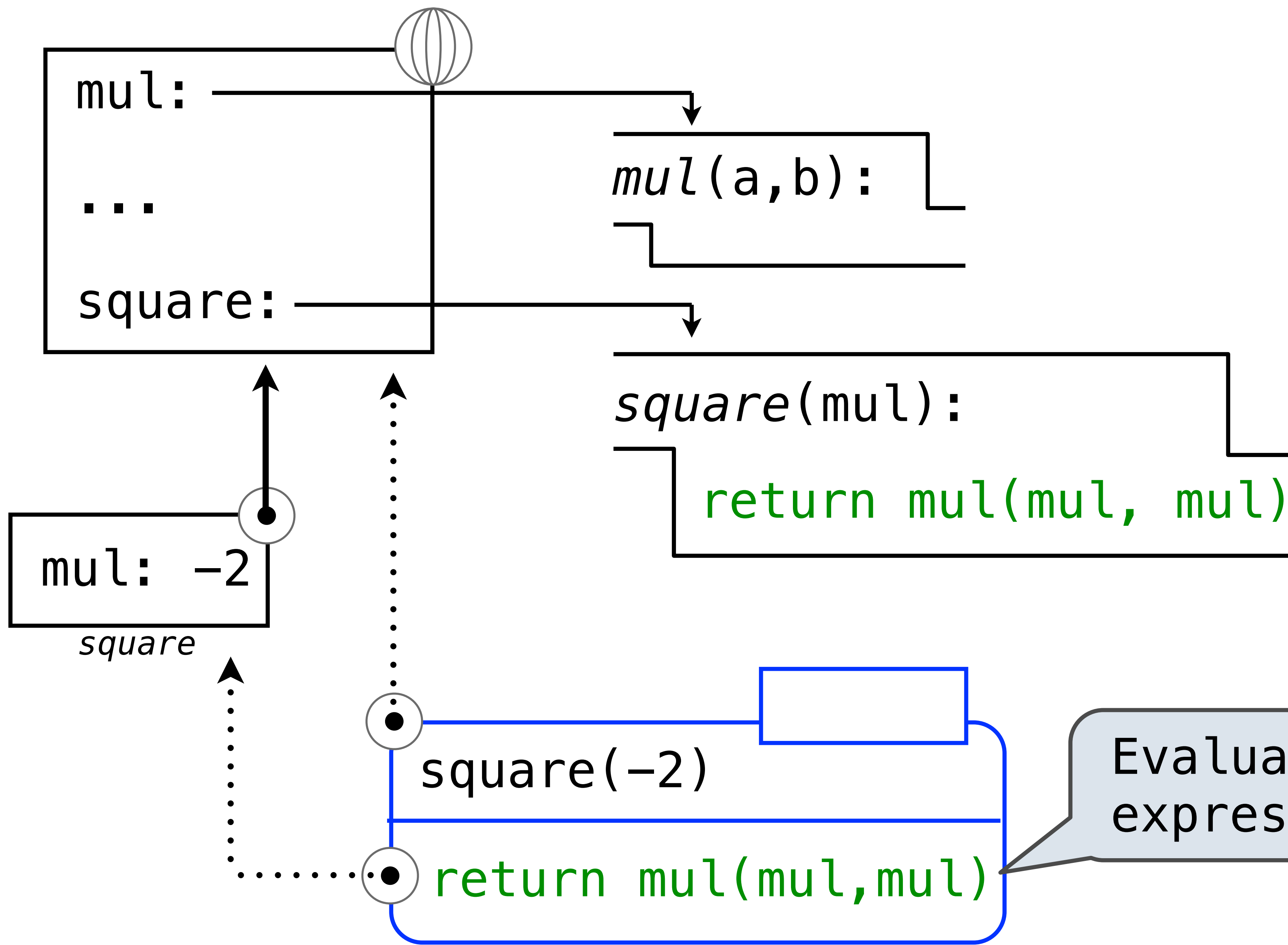


Shadowing Names

Careful!

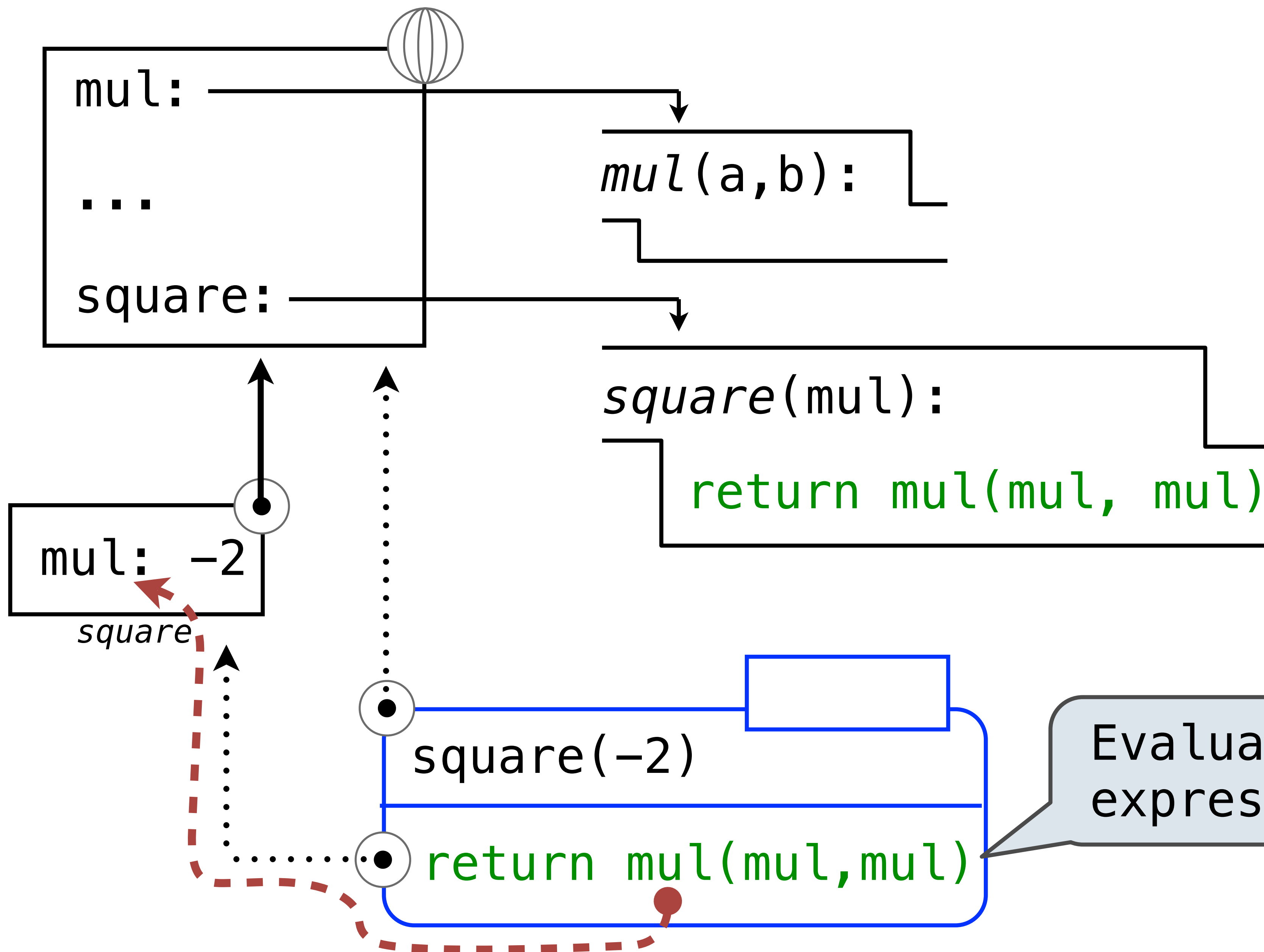
```
def square(mul):
    return mul(mul, mul)
square(-2)
```

If we use this formal parameter



Evaluating this call expression will fail

Shadowing Names



Python Feature Demonstration

<Demo>

Operators

Multiple Return Values

Docstrings

Doctests

Default Arguments

Statements

</Demo>

Statements

```
A statement
is executed by the interpret
to perform an action
```

Statements

```
A statement
is executed by the interpret
to perform an action
```

Compound statements:

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

Statements

A statement
is executed by the interpret
to perform an action

Compound statements:

Statement

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

Statements

A statement
is executed by the interpret
to perform an action

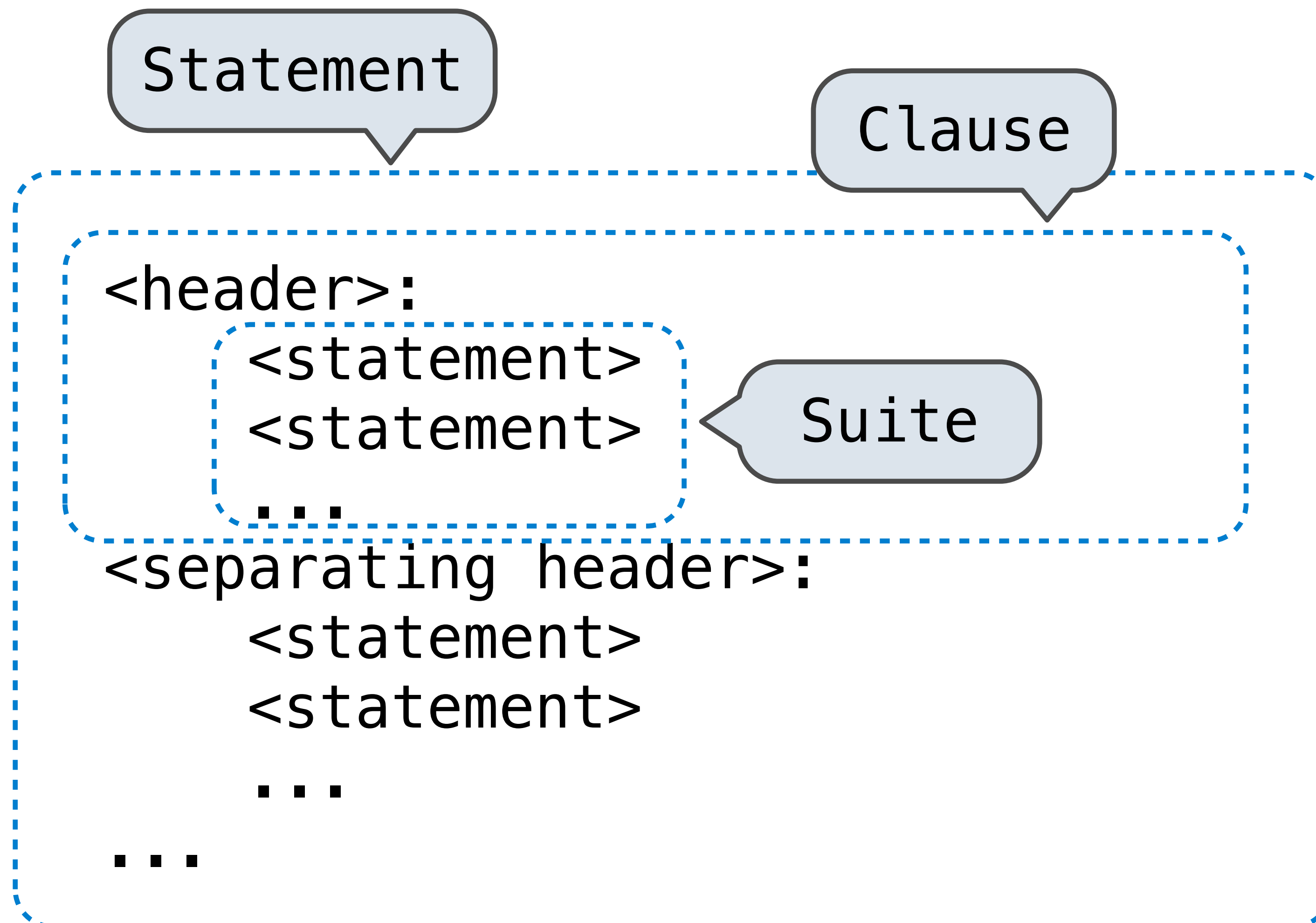
Compound statements:


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

Statements

A statement
is executed by the interpret
to perform an action

Compound statements:



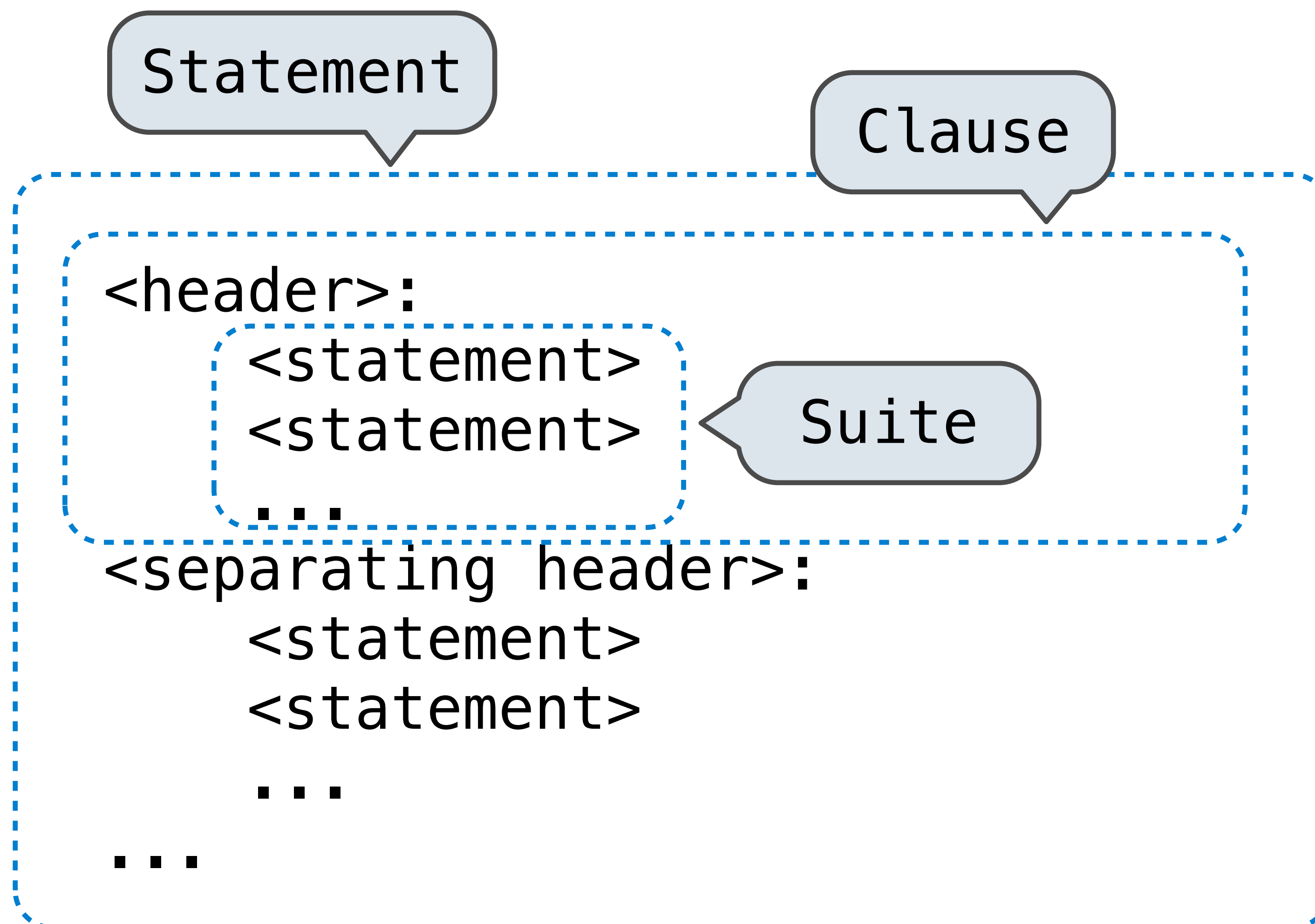
Statements

A statement

is executed by the interpret

to perform an action

Compound statements:



The first header determines a statement's type

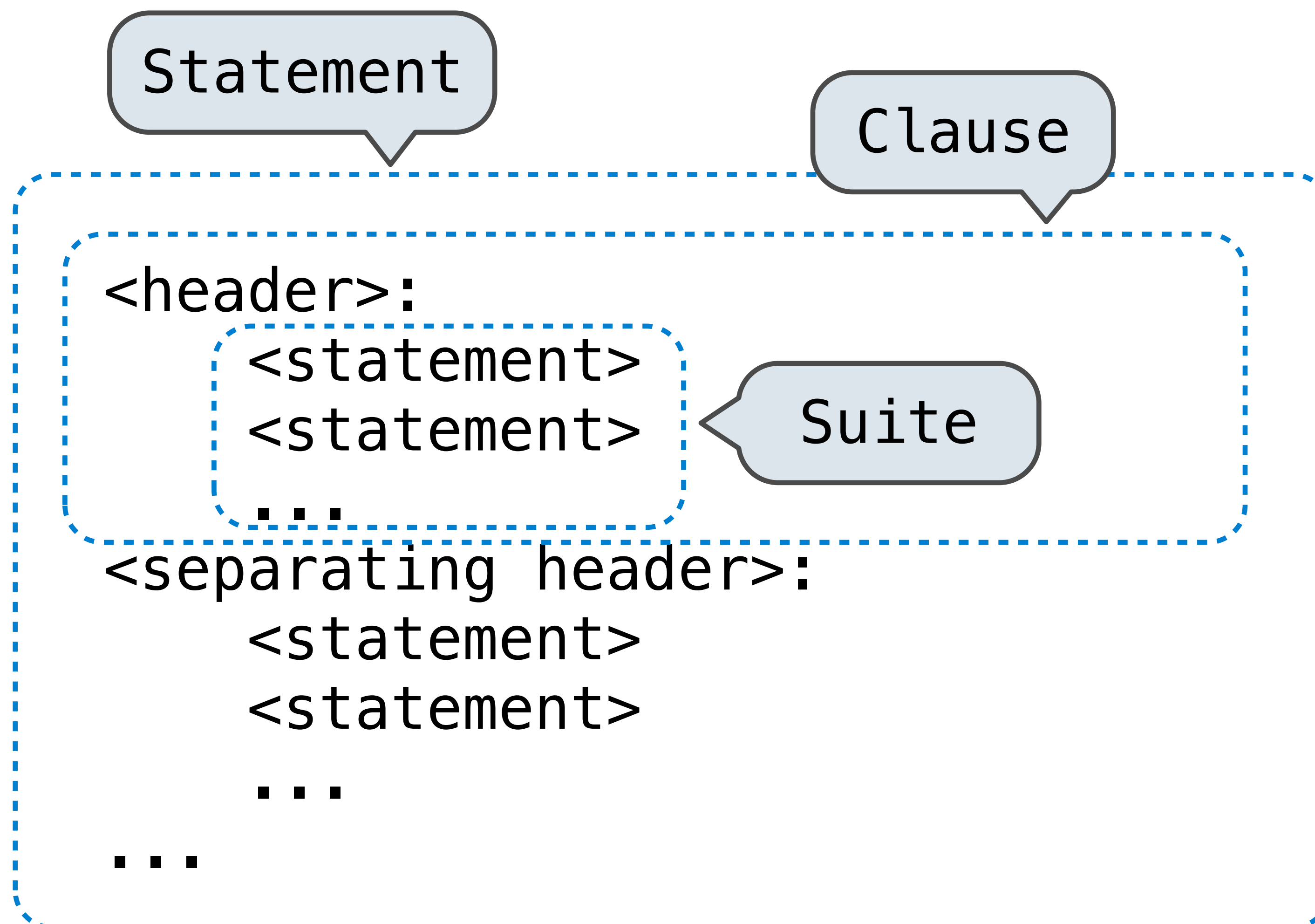
Statements

A statement

is executed by the interpret

to perform an action

Compound statements:



The first header determines a statement's type

The header of a clause "controls" the suite that follows

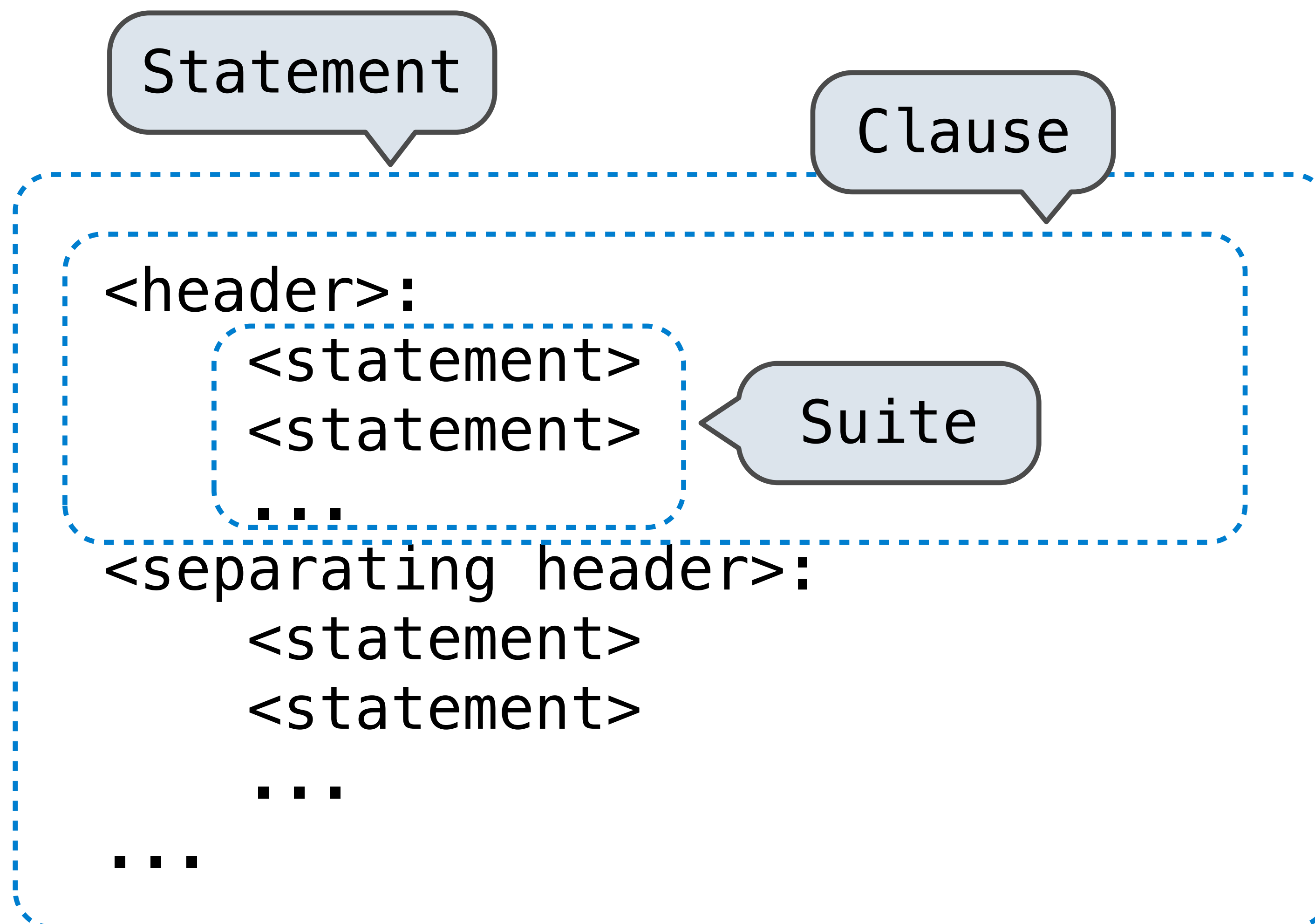
Statements

A statement

is executed by the interpret

to perform an action

Compound statements:



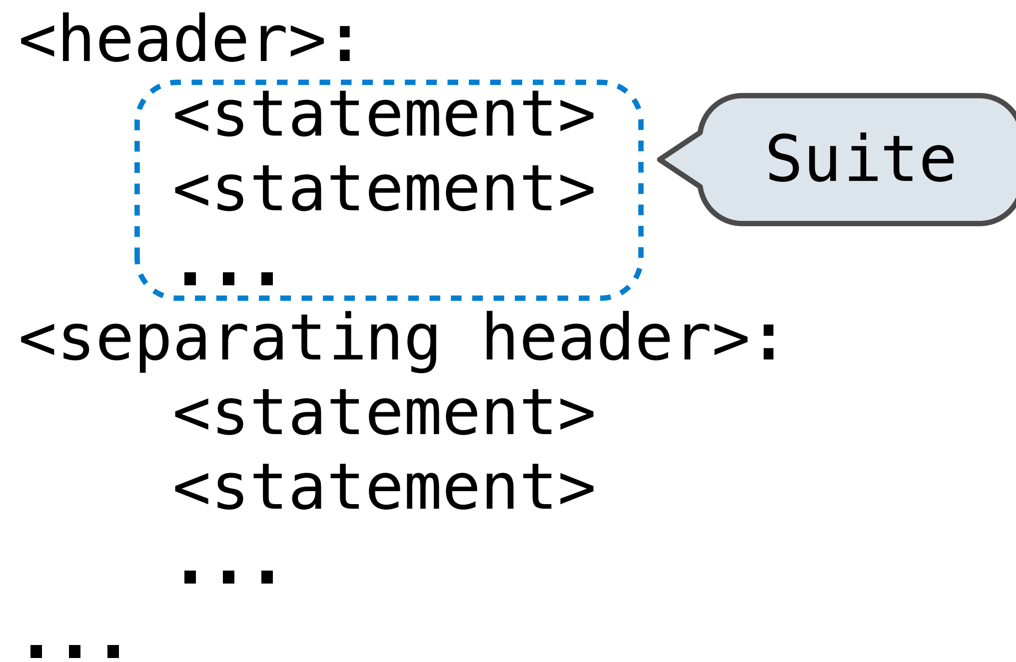
The first header determines a statement's type

The header of a clause "controls" the suite that follows

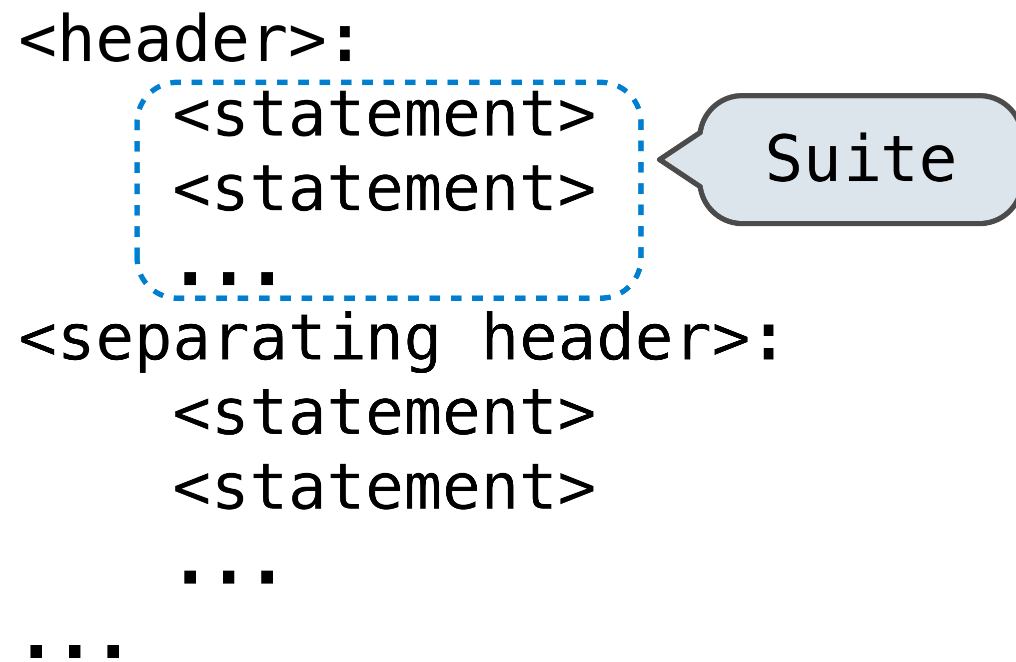
def statements are compound statements

Compound Statements

Compound statements:



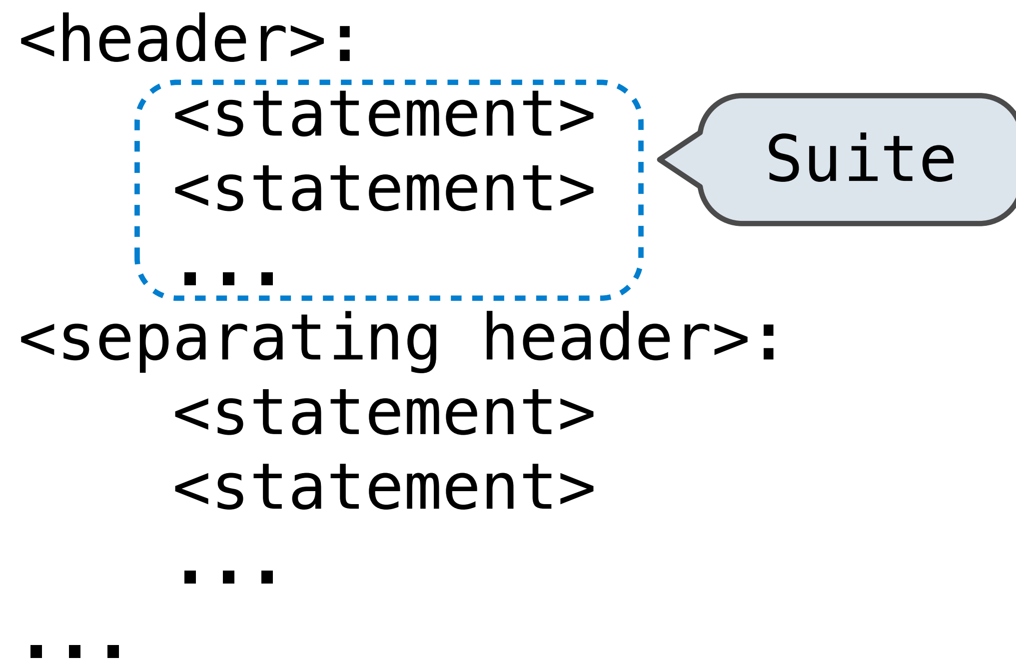
Compound Statements



A suite is a sequence of statements

Compound Statements

Compound statements:

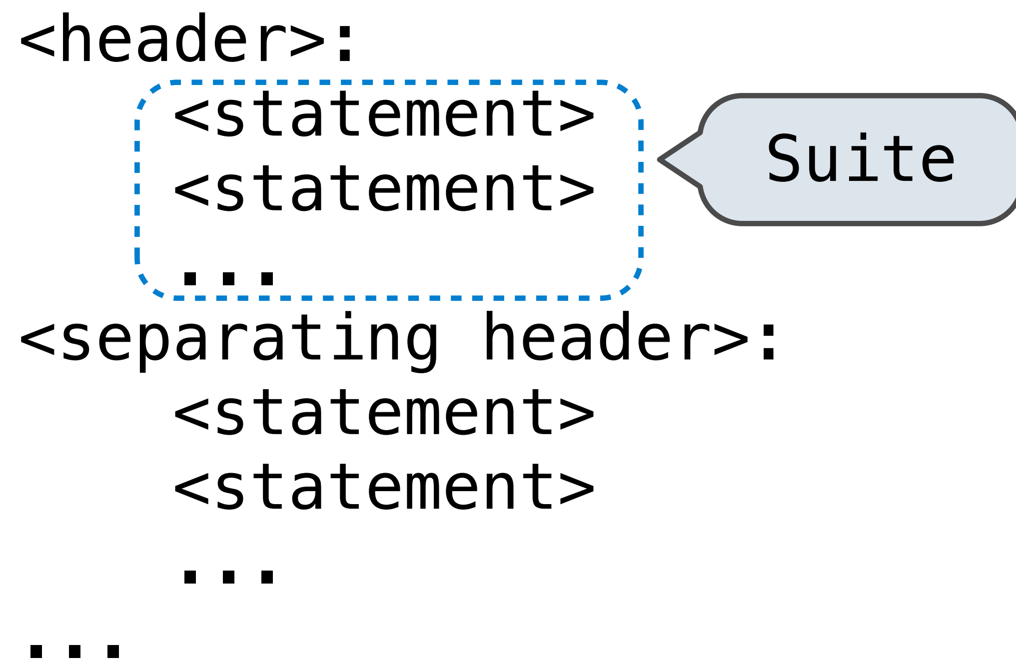


A suite is a sequence of statements

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

Compound Statements

Compound statements:



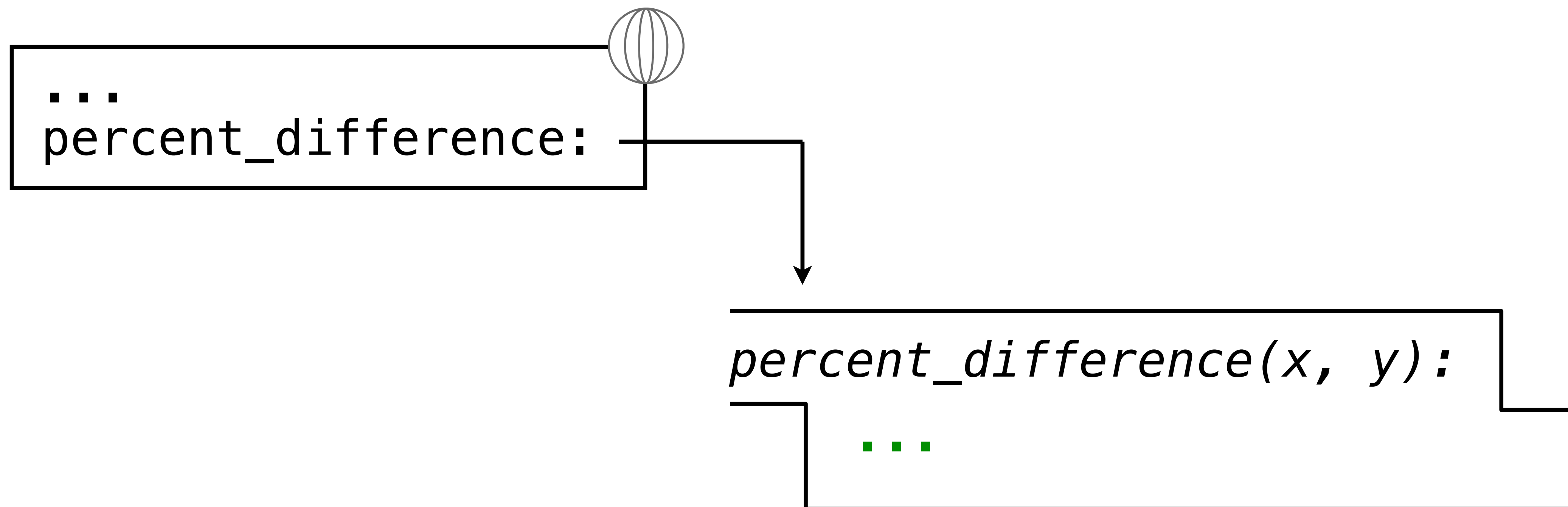
A suite is a sequence of statements

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

Execution Rule for a sequence of statements:

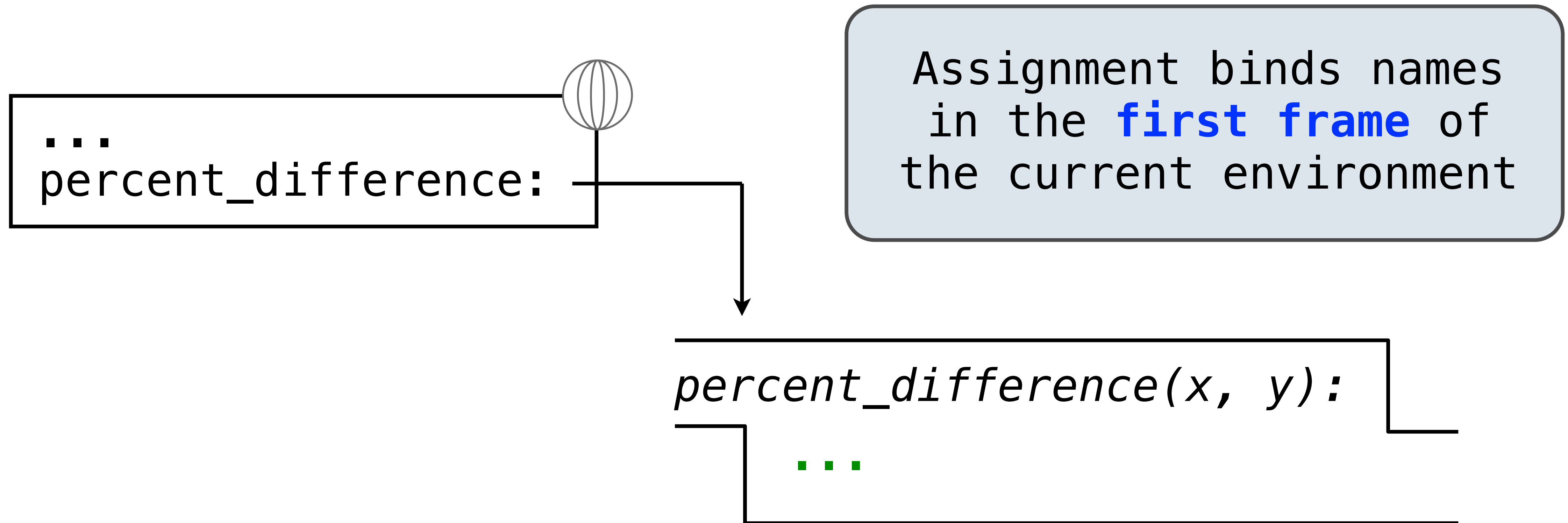
- Execute the first
- Unless directed otherwise, execute the rest

Local Assignment



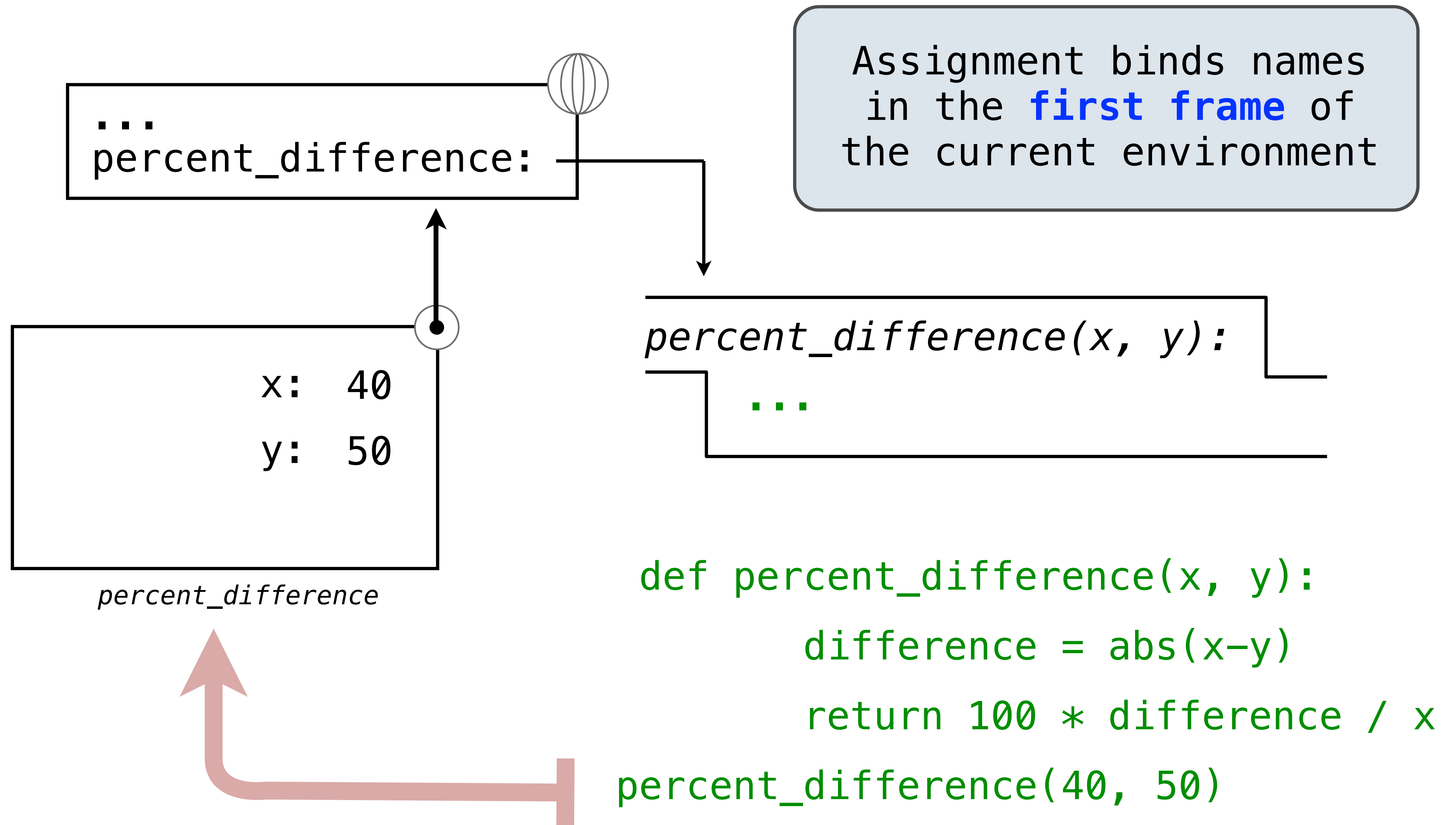

```
def percent_difference(x, y):
        difference = abs(x-y)
        return 100 * difference / x
percent_difference(40, 50)
```

Local Assignment

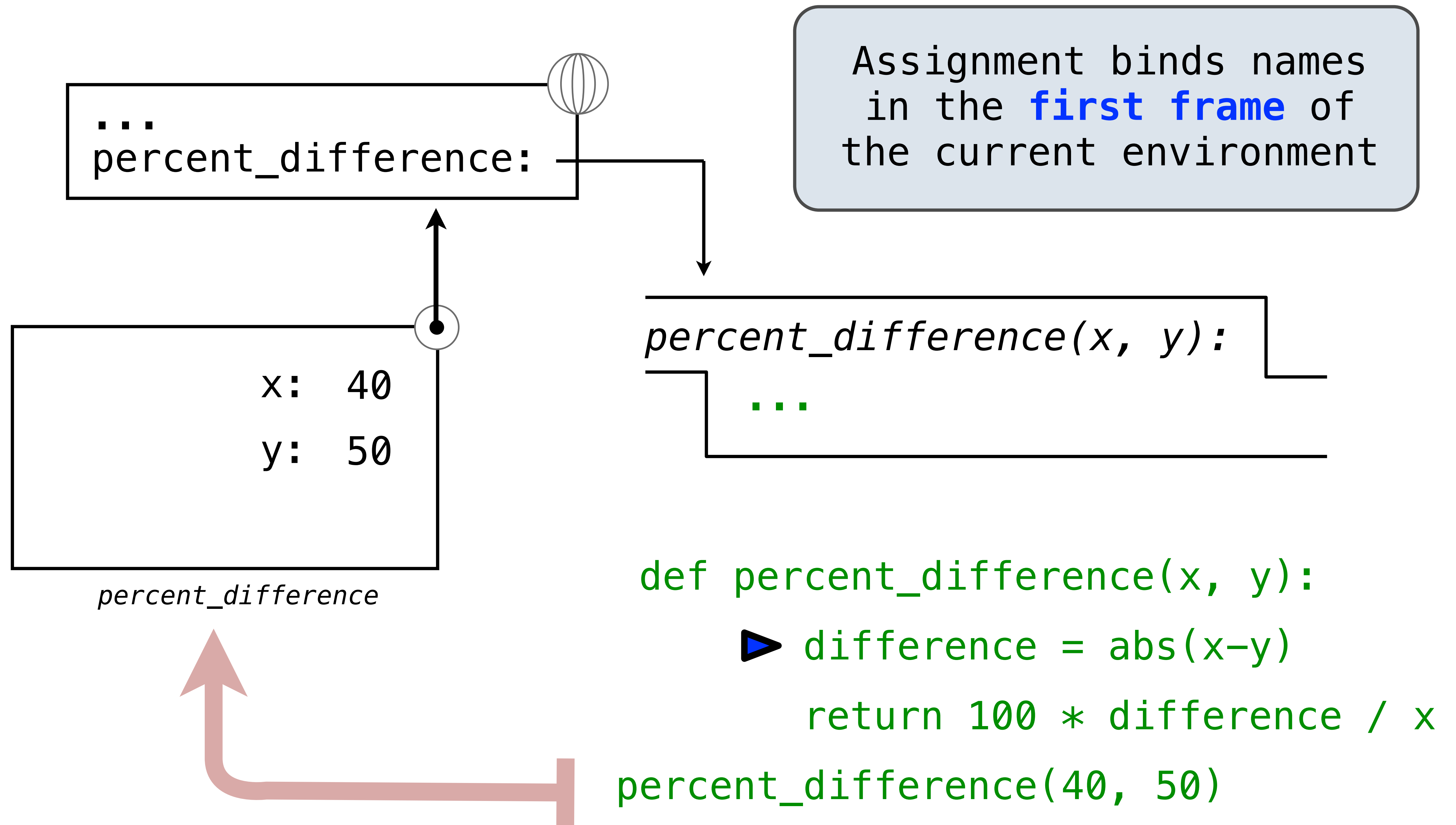



```
def percent_difference(x, y):
        difference = abs(x-y)
        return 100 * difference / x
percent_difference(40, 50)
```

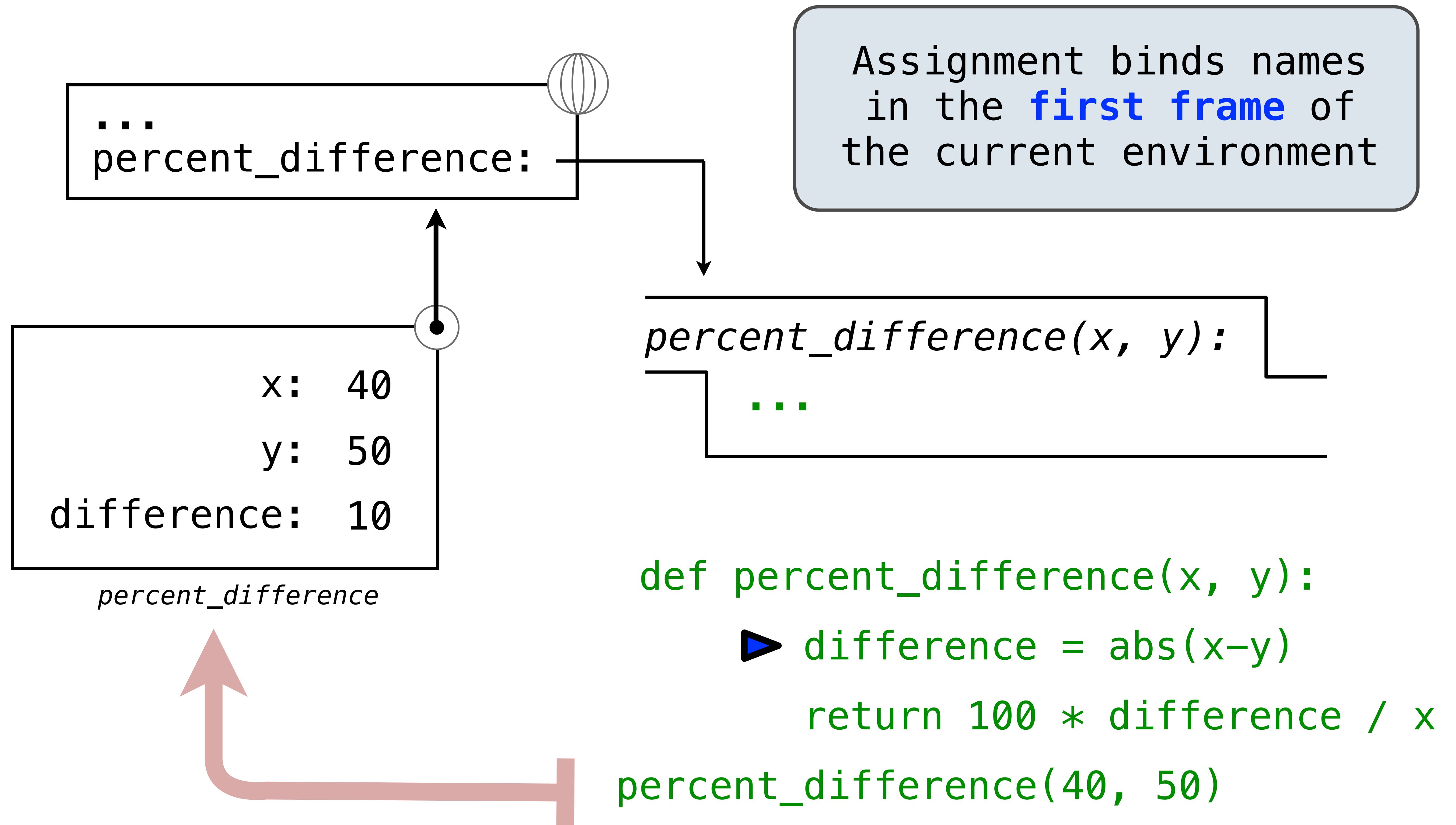
Local Assignment



Local Assignment



Local Assignment



Conditional Statements

```
def absolute_value(x):
        """Return the absolute value of x."""
        if x > 0:
            return x
        elif x == 0:
            return 0
        else:
            return -x
```

Conditional Statements

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
```

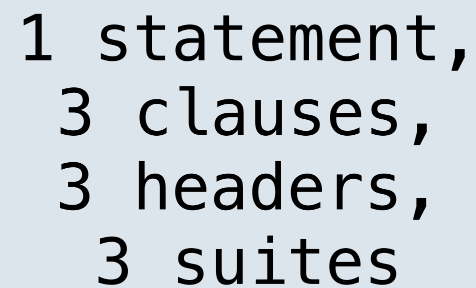
Conditional Statements

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x > 0:
        return x
    elif x == 0:
        return 0
    else:
        return -x
```

1 statement,
3 clauses,
3 headers,
3 suites

Execution rule for conditional statements:

Conditional Statements




```
def absolute_value(x):
        """Return the absolute value of x."""
        if x > 0:
            return x
        elif x == 0:
            return 0
        else:
            return -x
```

Execution rule for conditional statements:

Each clause is considered in order.

1. Evaluate the header's expression.

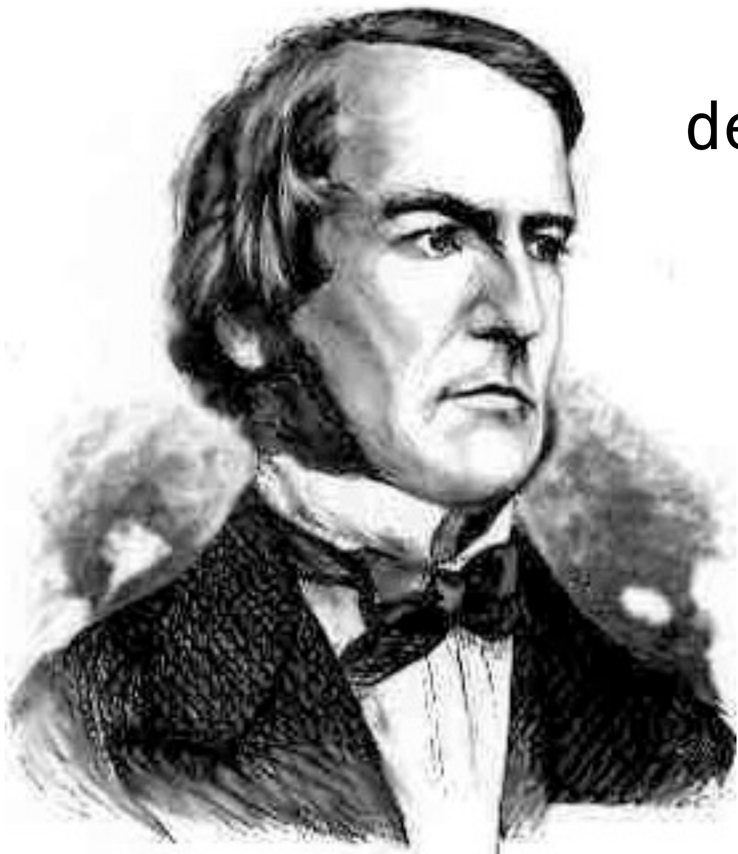
2. If it is a true value, execute the suite & skip the rest.

Boolean Contexts

```
def absolute_value(x):
        """Return the absolute value of x."""
        if x > 0:
            return x
        elif x == 0:
            return 0
        else:
            return -x
```

George Boole

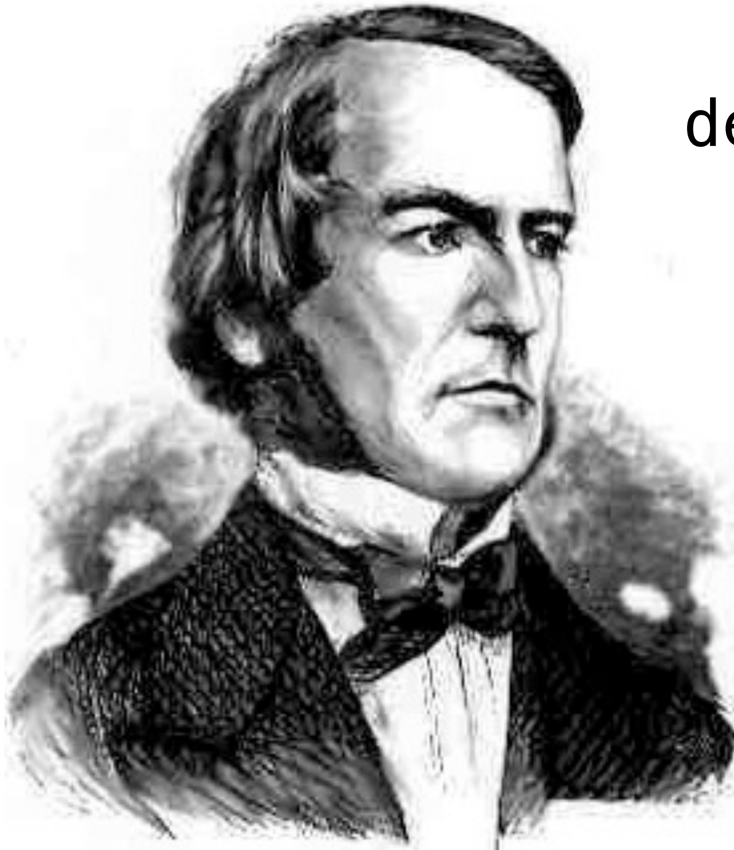
Boolean Contexts



George Boole

```
def absolute_value(x):
        """Return the absolute value of x."""
        if x > 0:
            return x
        elif x == 0:
            return 0
        else:
            return -x
```

Boolean Contexts

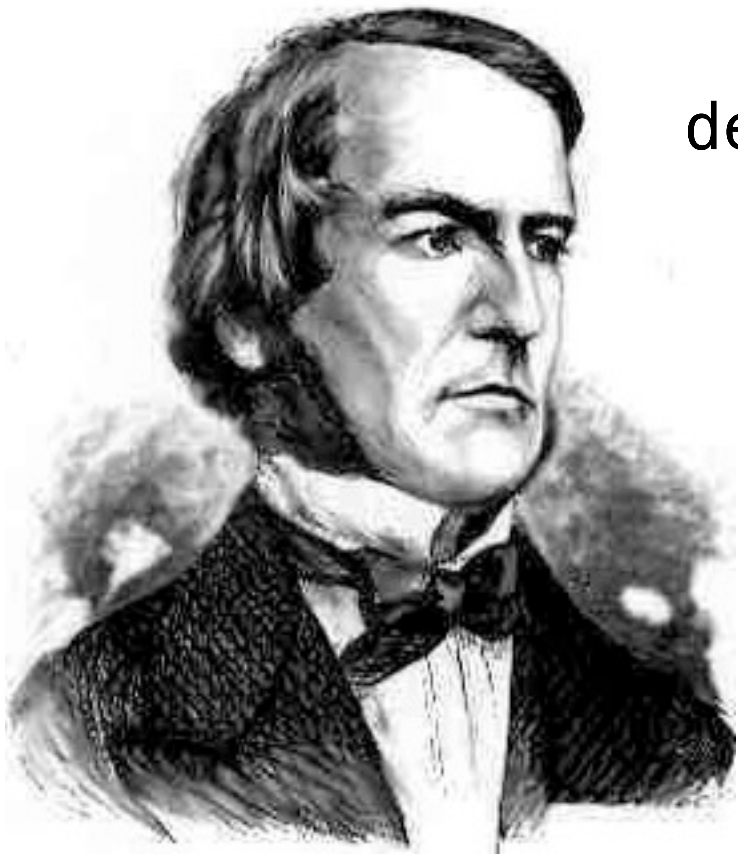


George Boole

```
def absolute_value(x):
        """Return the absolute value of x."""
        if x > 0:
            return x
        elif x == 0:
            return 0
        else:
            return -x
```

Two boolean contexts

Boolean Contexts

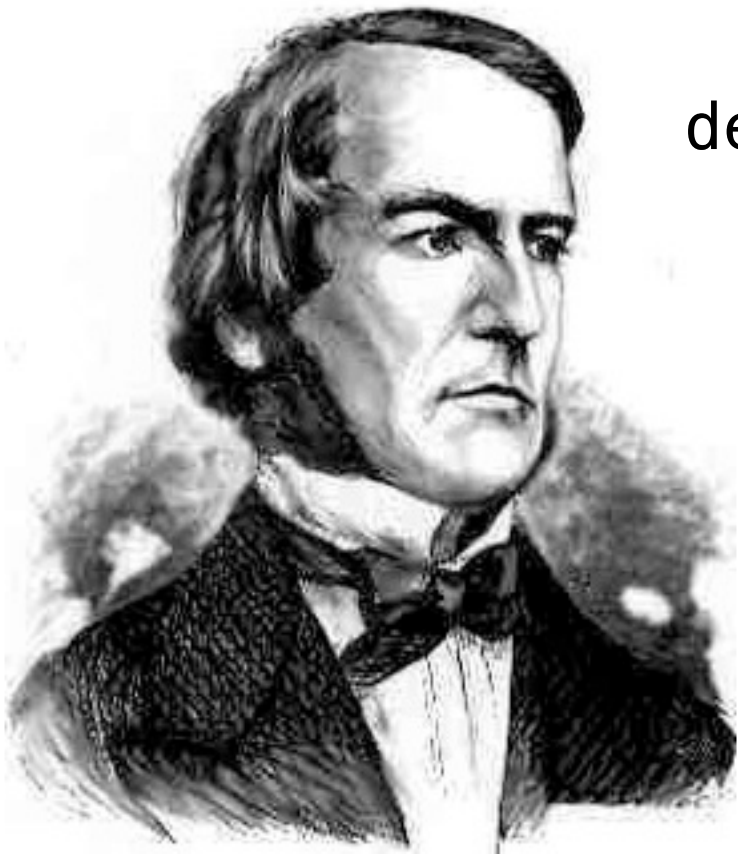


George Boole

```
def absolute_value(x):
        """Return the absolute value of x."""
        if x > 0:
            return x
        elif x == 0:
            return 0
        else:
            return -x
```

Two boolean contexts

Boolean Contexts

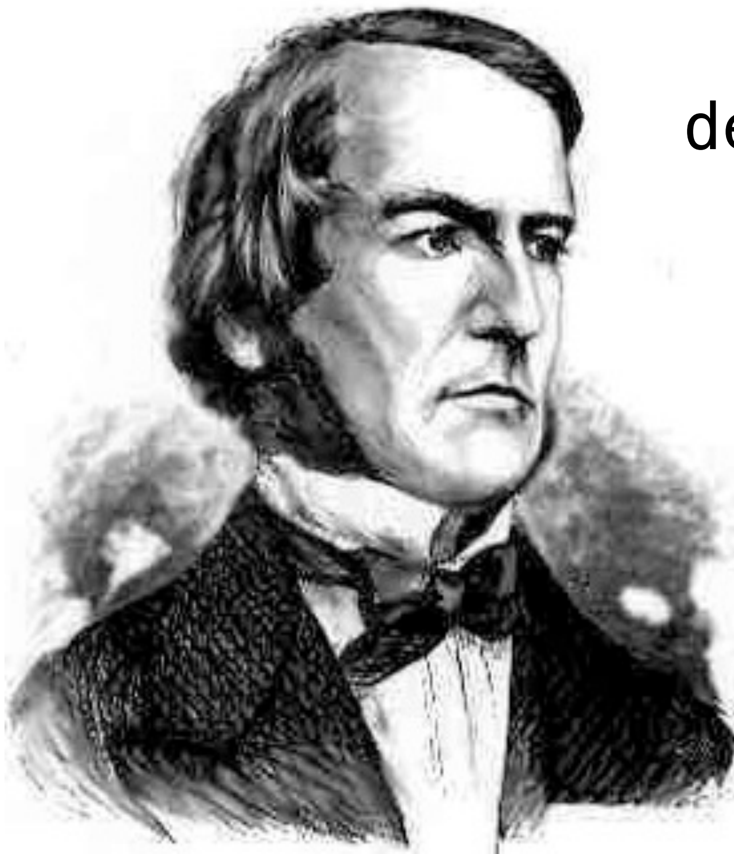


George Boole

```
def absolute_value(x):
        """Return the absolute value of x."""
        if x > 0:
            return x
        elif x == 0:
            return 0
        else:
            return -x
```

Two boolean contexts

Boolean Contexts



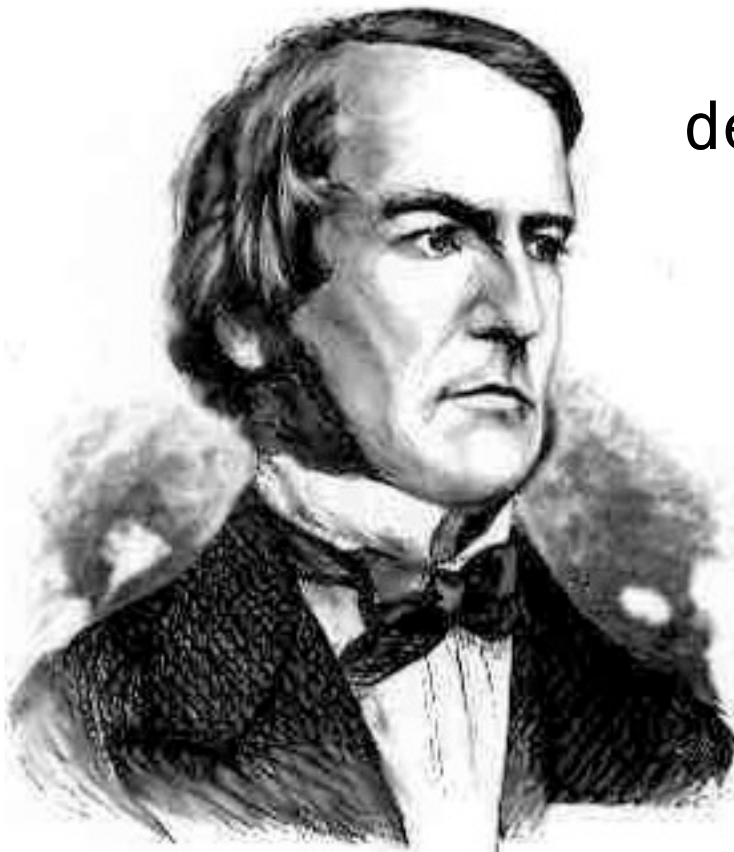
George Boole

```
def absolute_value(x):
        """Return the absolute value of x."""
        if x > 0:
            return x
        elif x == 0:
            return 0
        else:
            return -x
```

Two boolean contexts

False values in Python: False, 0, ‘’, None

Boolean Contexts



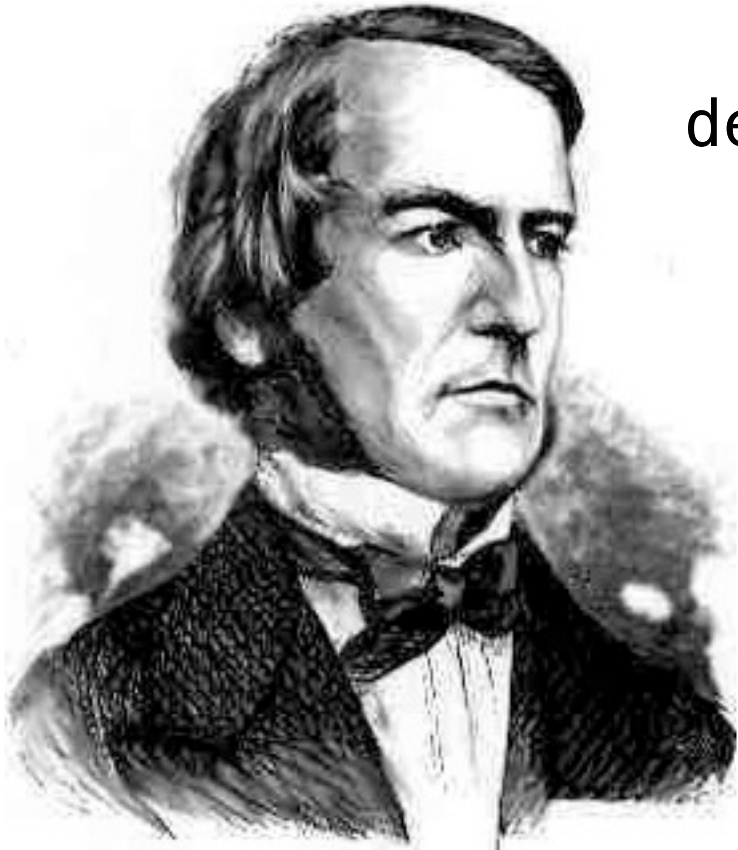
George Boole

```
def absolute_value(x):
        """Return the absolute value of x."""
        if x > 0:
            return x
        elif x == 0:
            return 0
        else:
            return -x
```

Two boolean contexts

False values in Python: False, 0, ‘’, None *(more to come)*

Boolean Contexts



George Boole

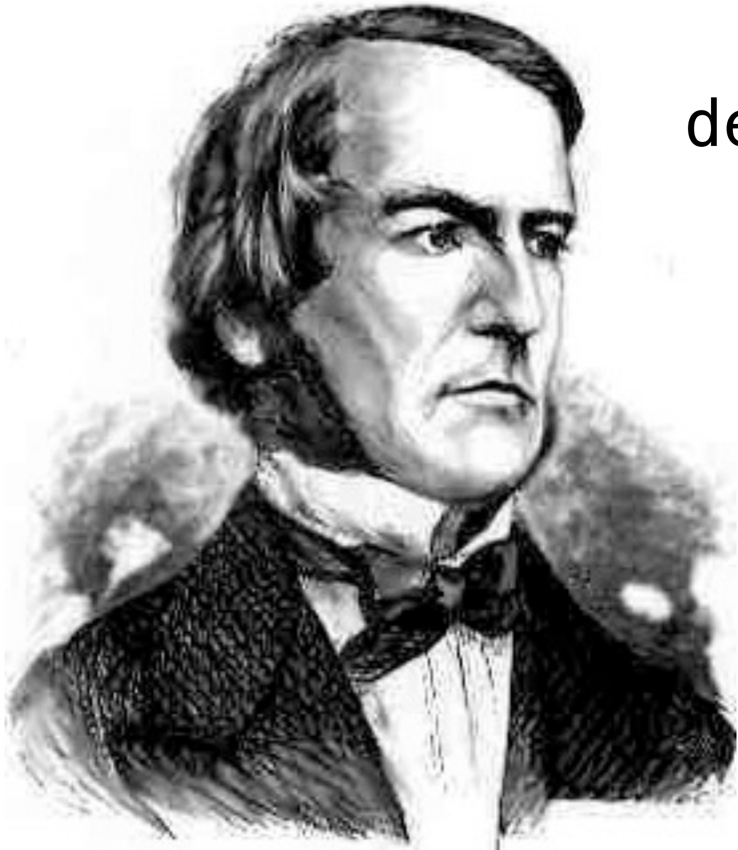
```
def absolute_value(x):
        """Return the absolute value of x."""
        if x > 0:
            return x
        elif x == 0:
            return 0
        else:
            return -x
```

Two boolean contexts

False values in Python: False, 0, ‘’, None *(more to come)*

True values in Python: Anything else (True)

Boolean Contexts



George Boole

```
def absolute_value(x):
        """Return the absolute value of x."""
        if x > 0:
            return x
        elif x == 0:
            return 0
        else:
            return -x
```

Two boolean contexts

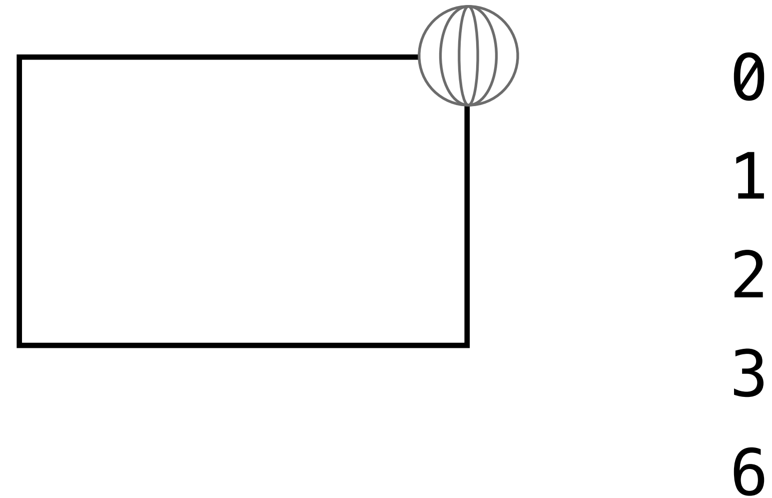
False values in Python: False, 0, ‘’, None *(more to come)*

True values in Python: Anything else (True)

Read Section 1.5.4!

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



0
1
2
3
6

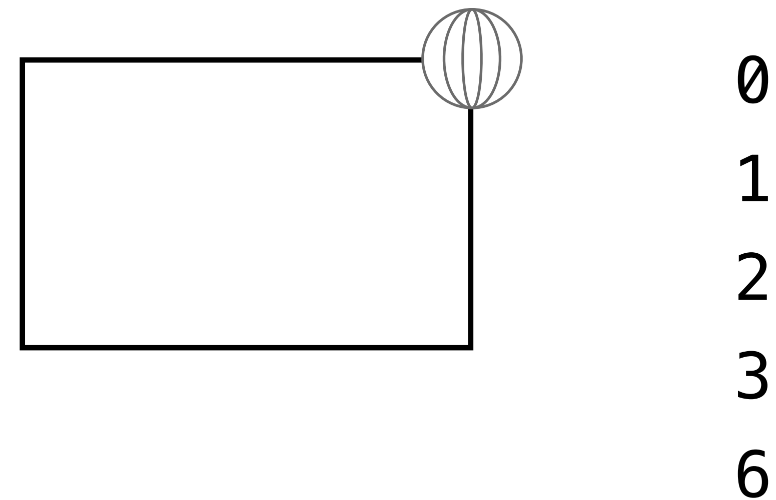
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



0
1
2
3
6

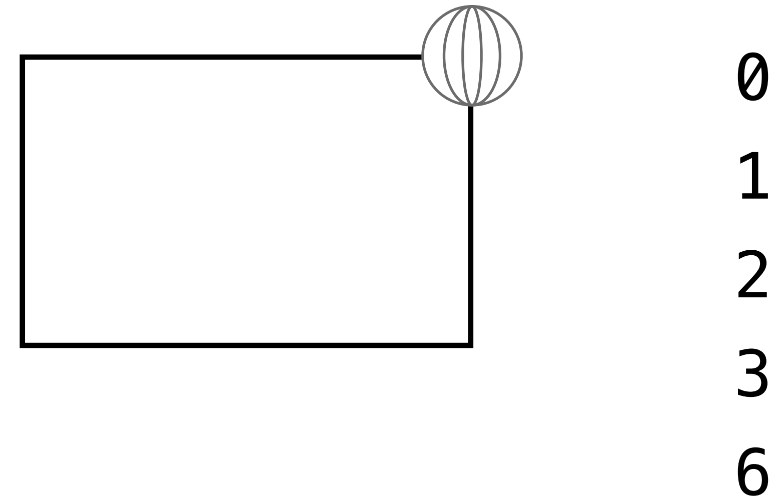
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



0
1
2
3
6

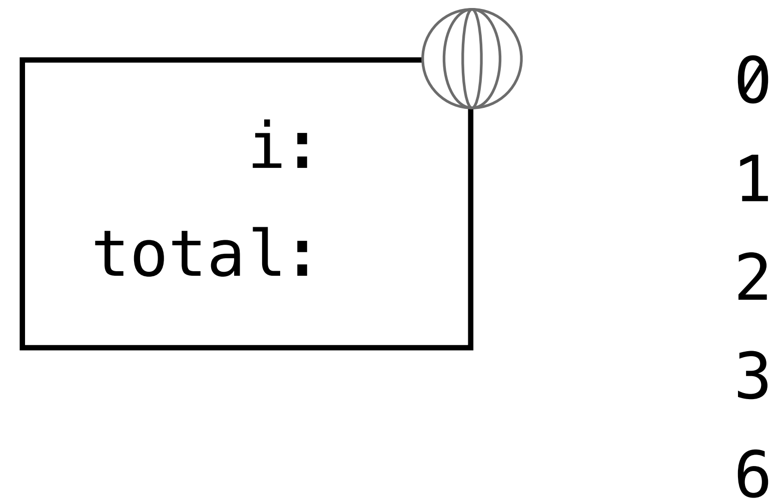
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
▶ i, total = 0, 0
  while i < 3:
      i = i + 1
      total = total + i
```



0
1
2
3
6

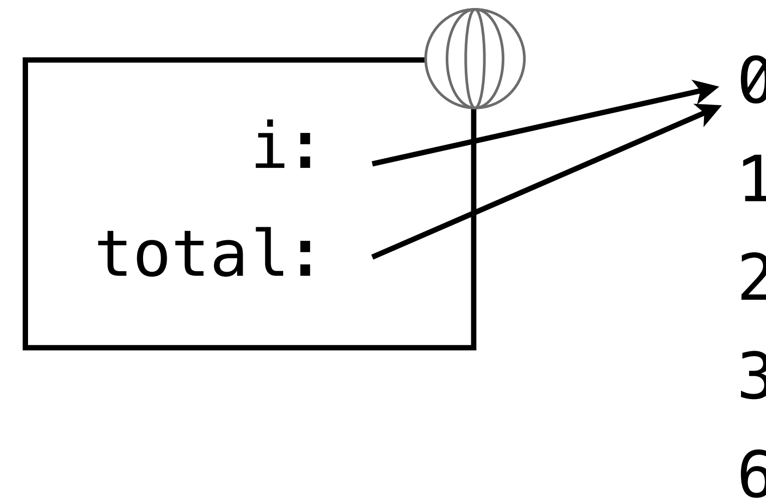
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



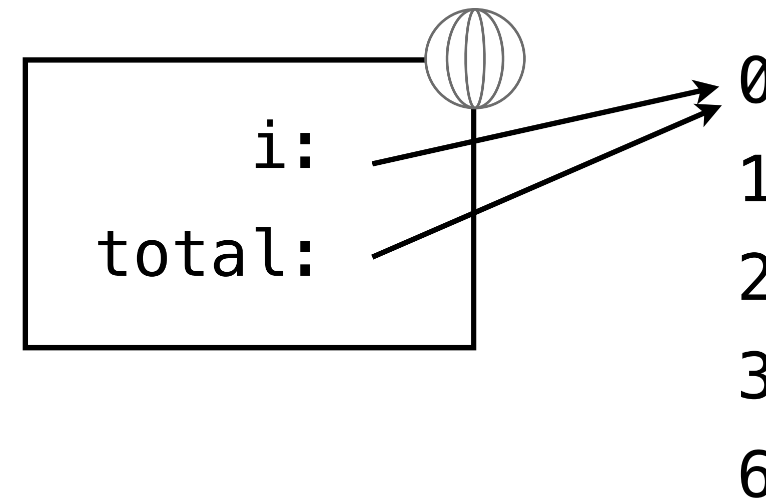
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



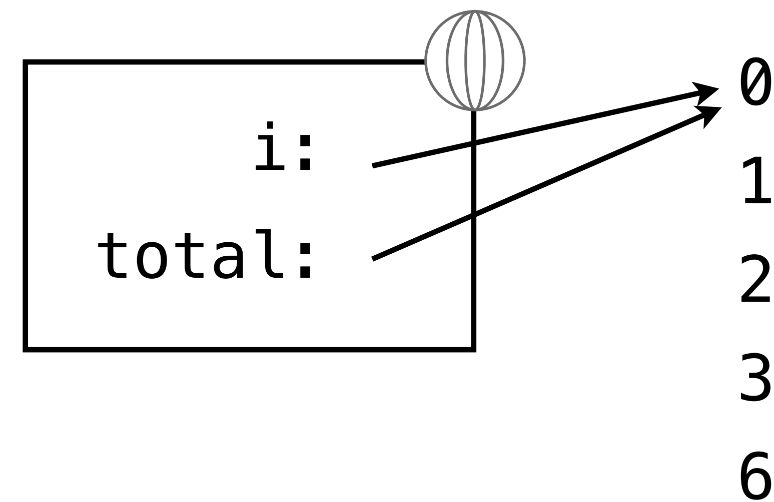
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

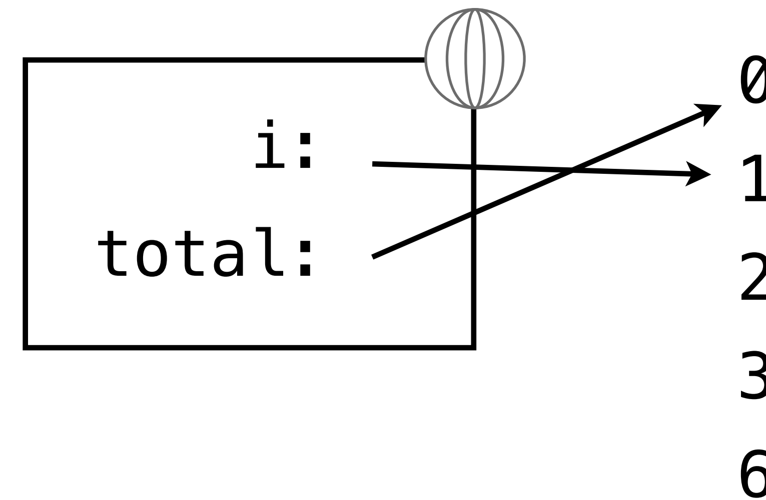


Execution rule for while statements:

1. Evaluate the header's expression.
2. If it is a true value, execute the (*whole*) suite, then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



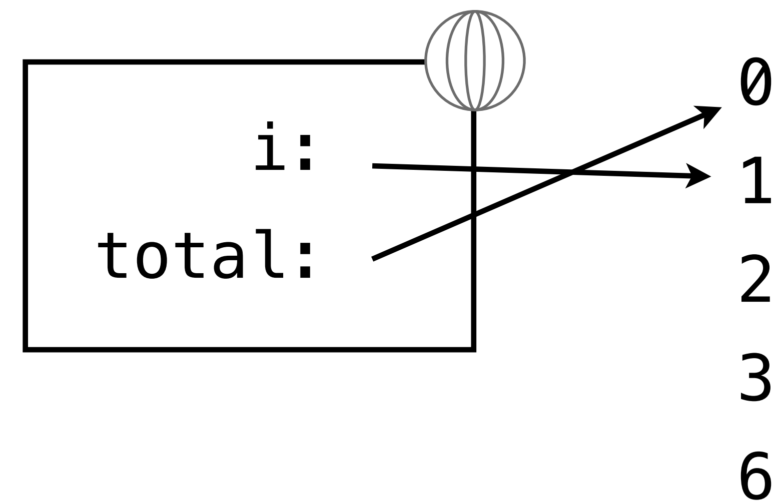
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



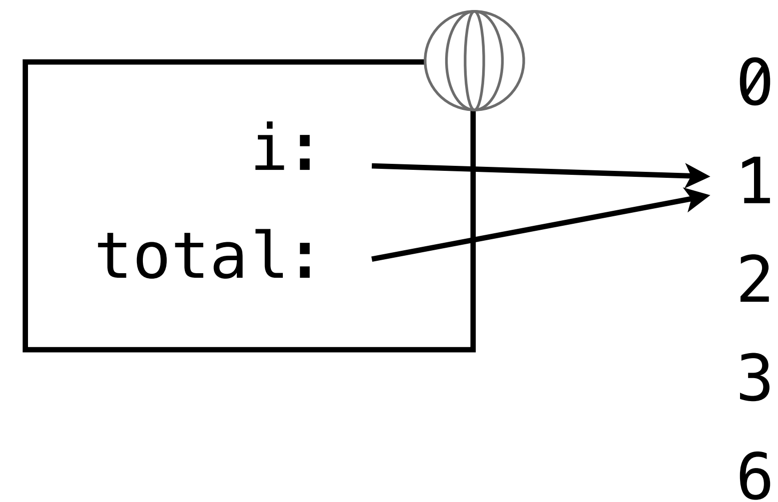
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



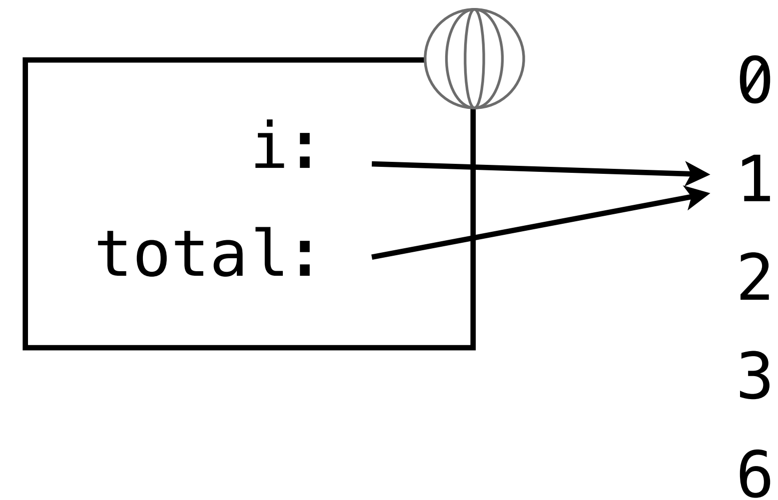
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



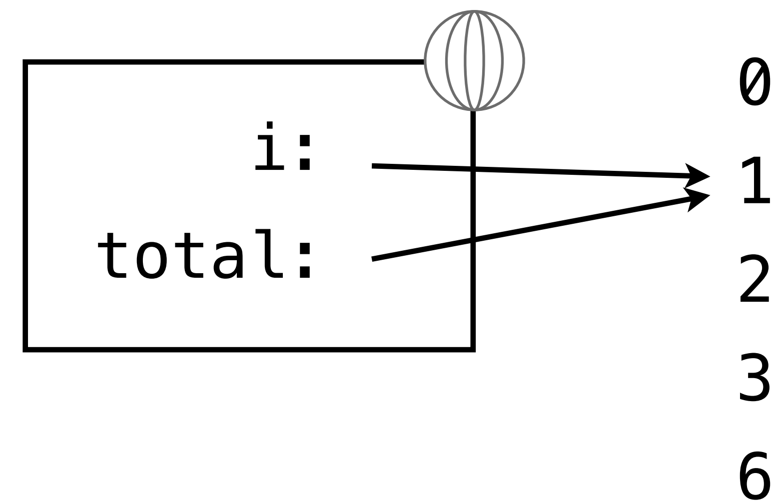
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



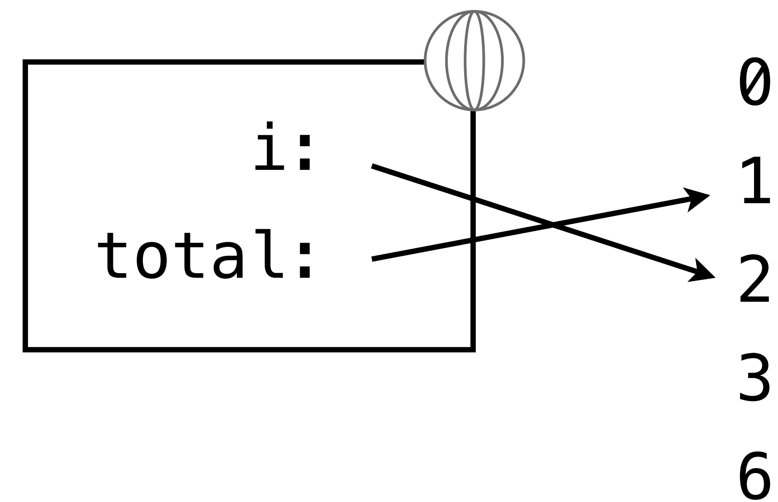
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



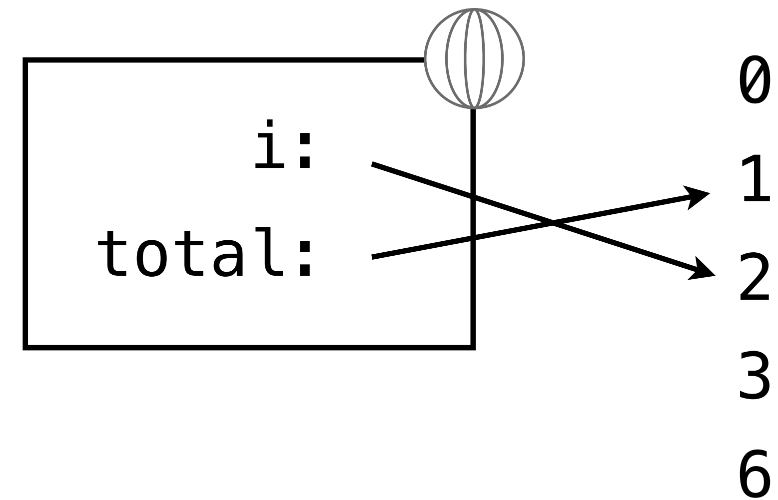
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



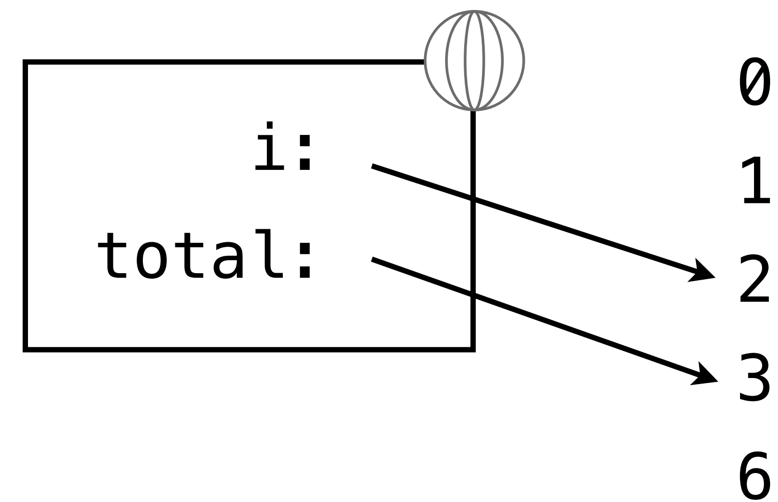
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



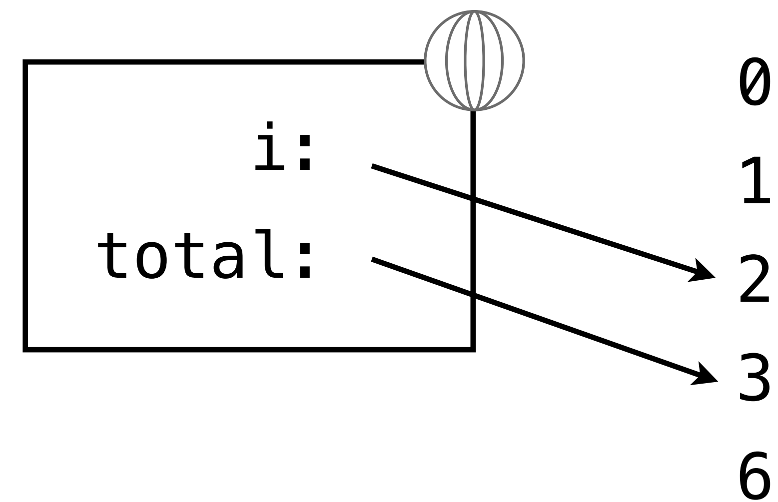
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

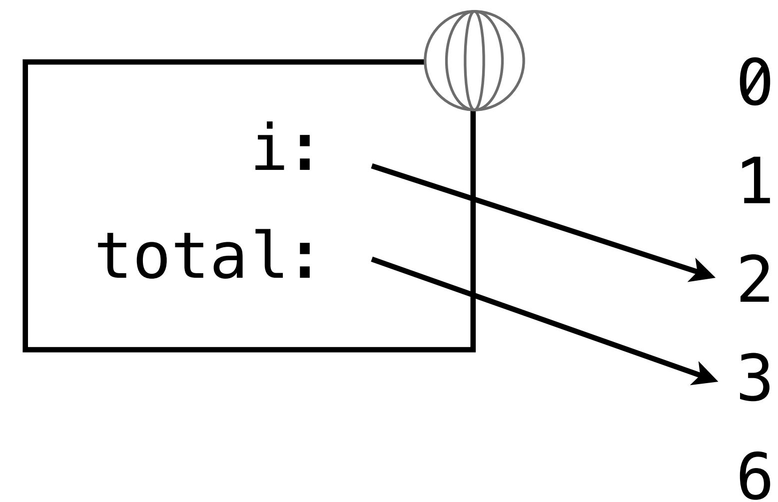


Execution rule for while statements:

1. Evaluate the header's expression.
2. If it is a true value, execute the (*whole*) suite, then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



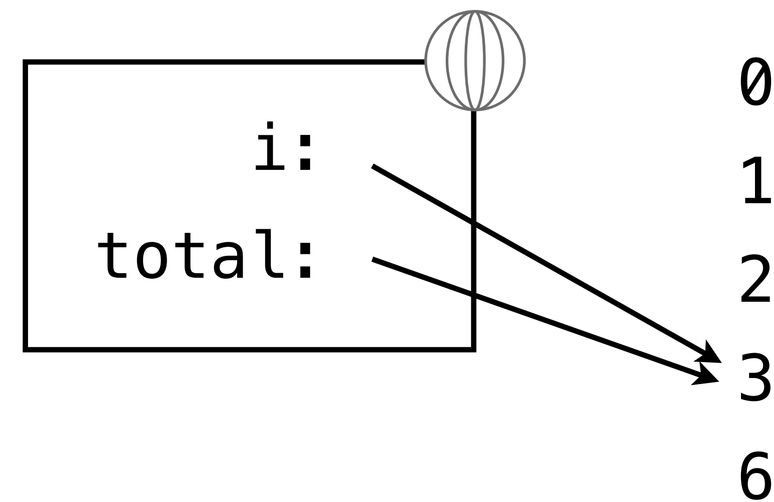
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



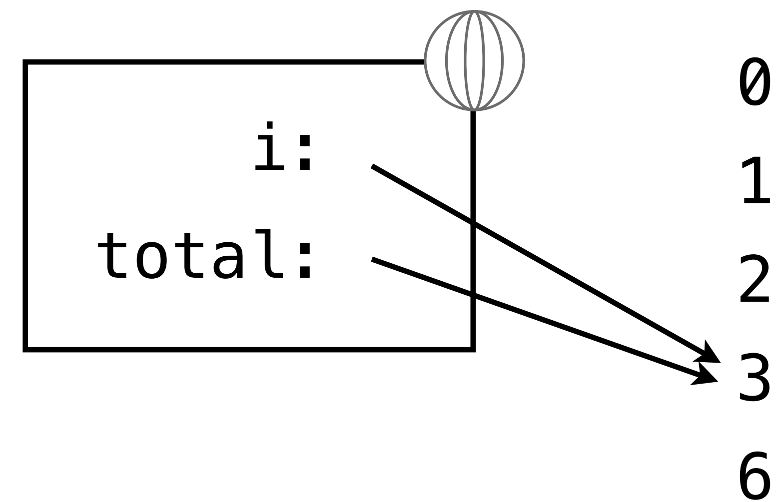
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



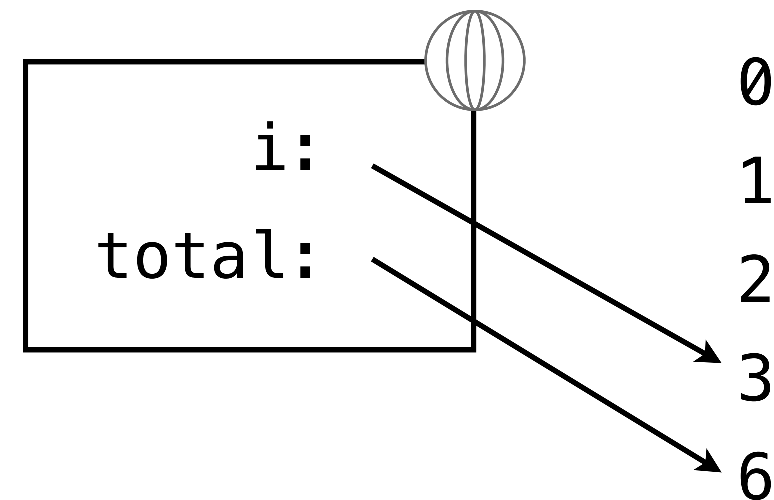
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



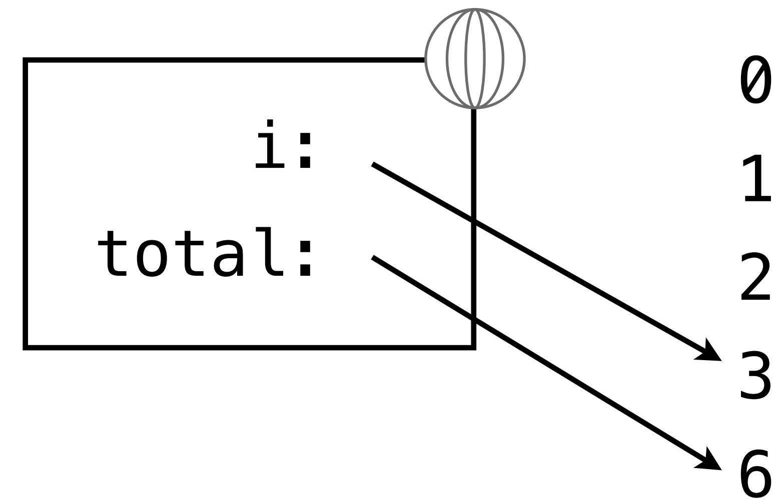
Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Iteration

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



Execution rule for while statements:

1. Evaluate the header's expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

The Fibonacci Sequence

The Fibonacci Sequence

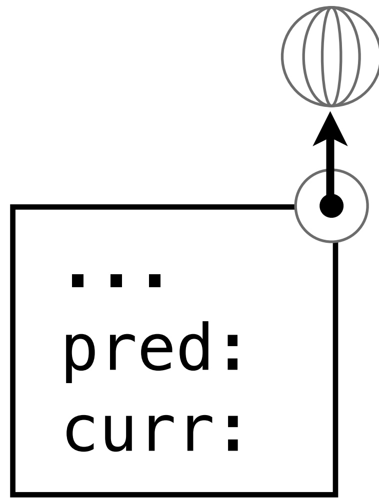
0, 1, 1, 2, 3, 5, 8, 13, ...

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
```

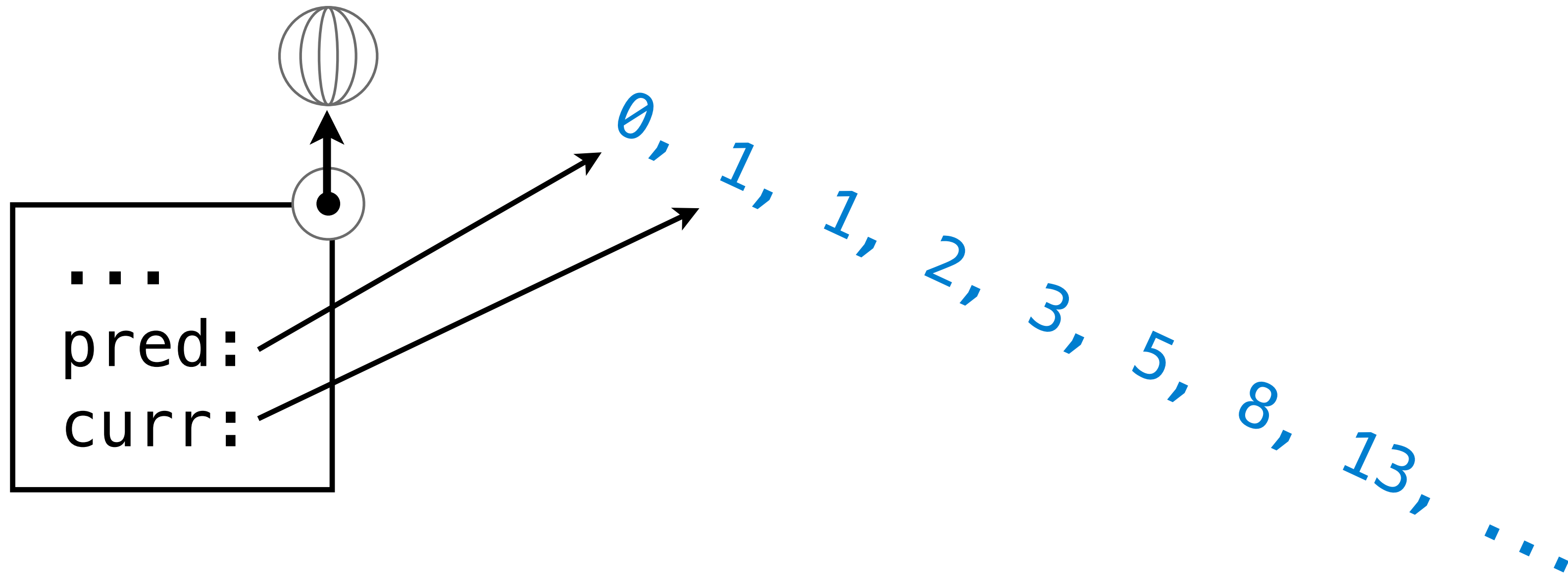
The Fibonacci Sequence



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

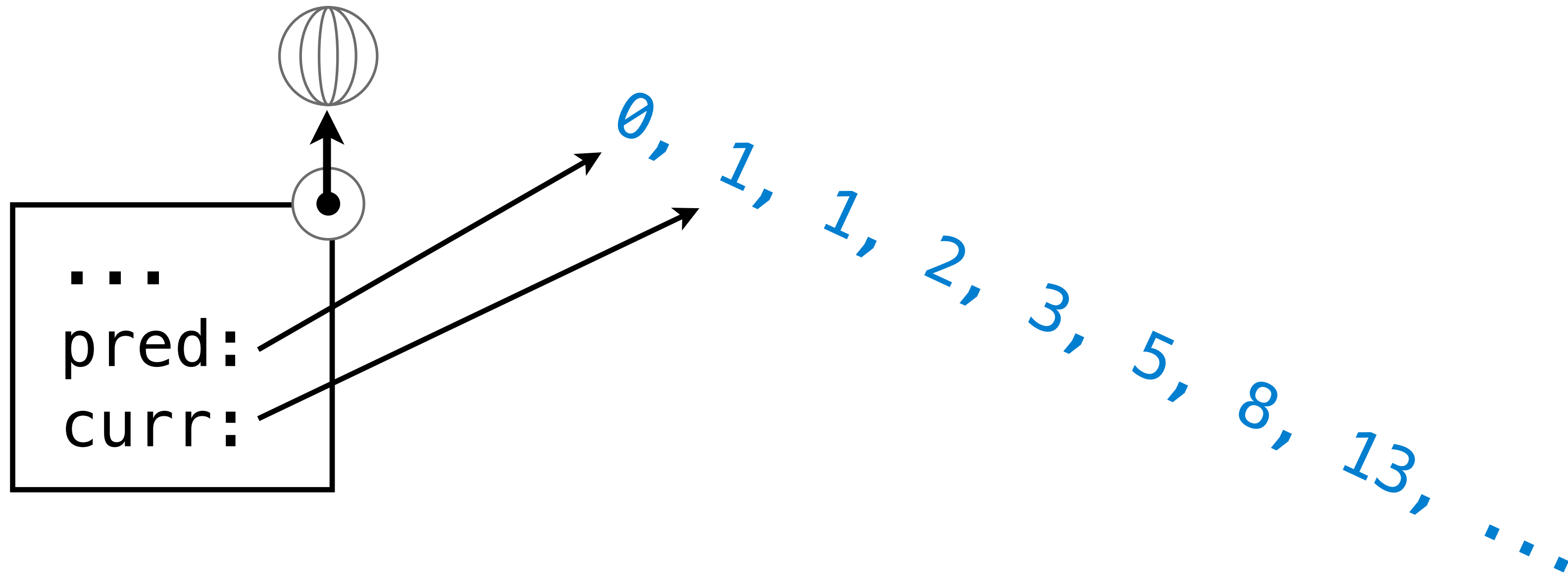
```
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1   # First two Fibonacci numbers
    k = 2               # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

The Fibonacci Sequence



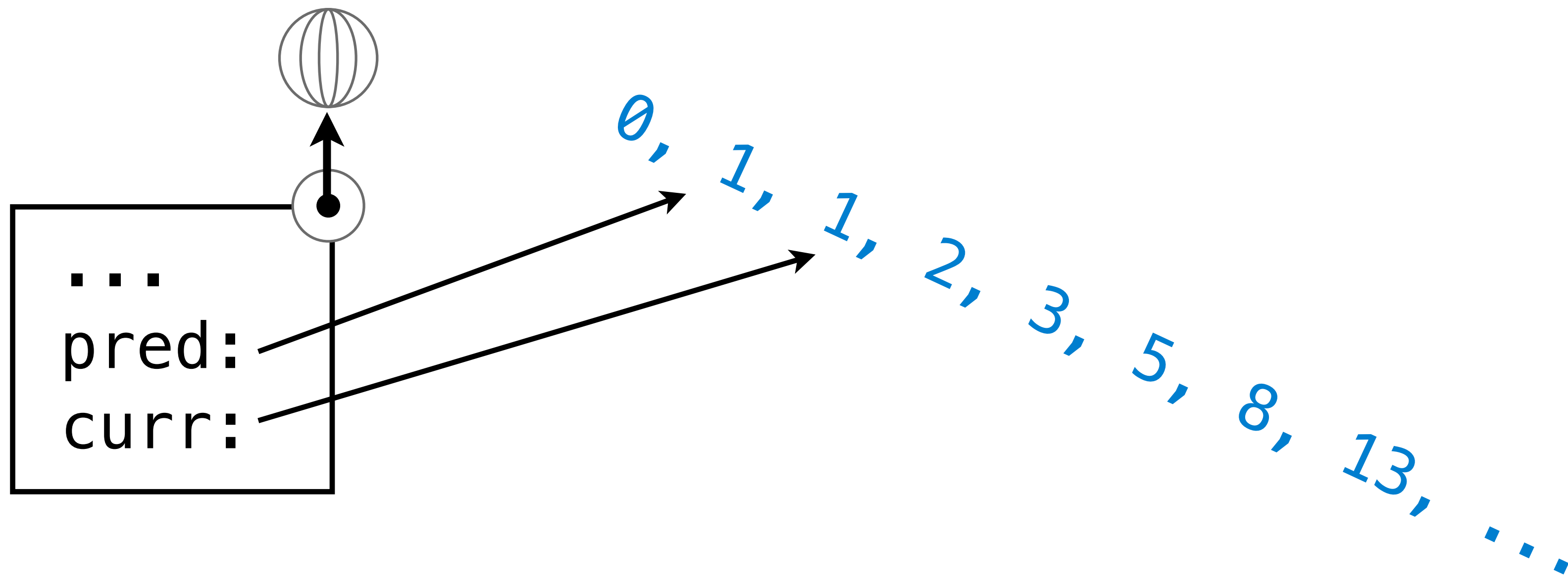

```
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1   # First two Fibonacci numbers
    k = 2               # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

The Fibonacci Sequence



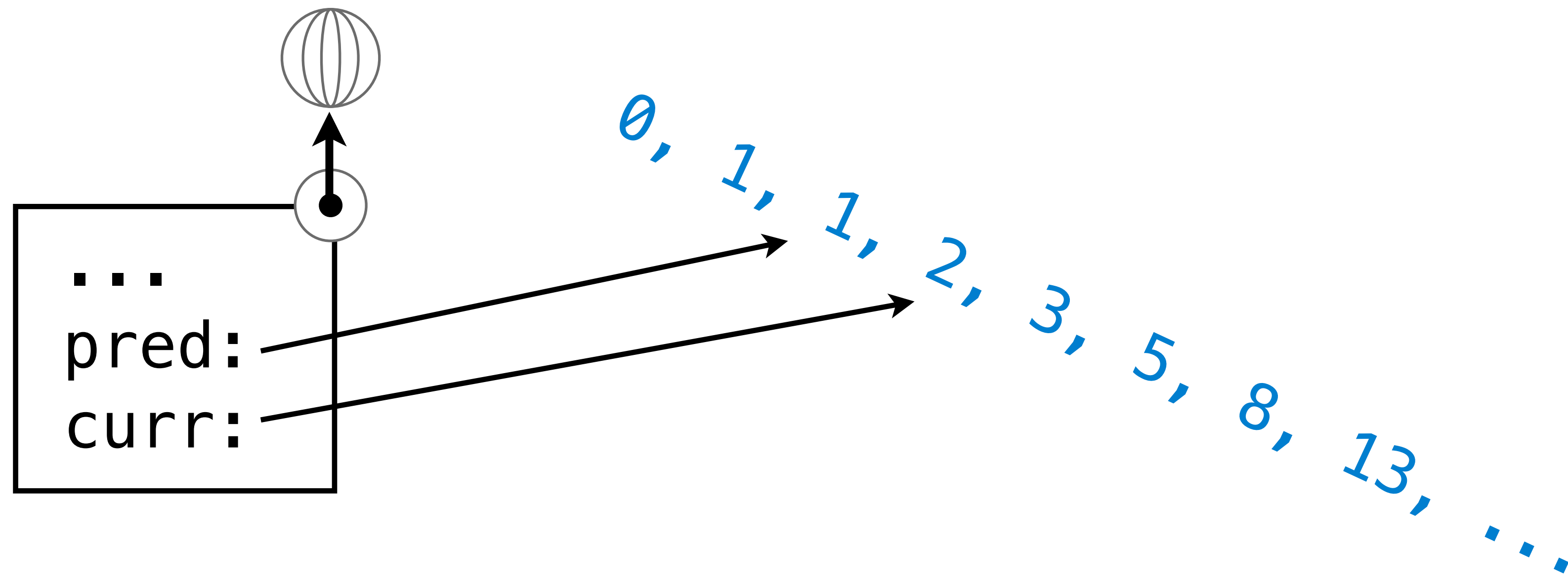

```
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1   # First two Fibonacci numbers
    k = 2               # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

The Fibonacci Sequence



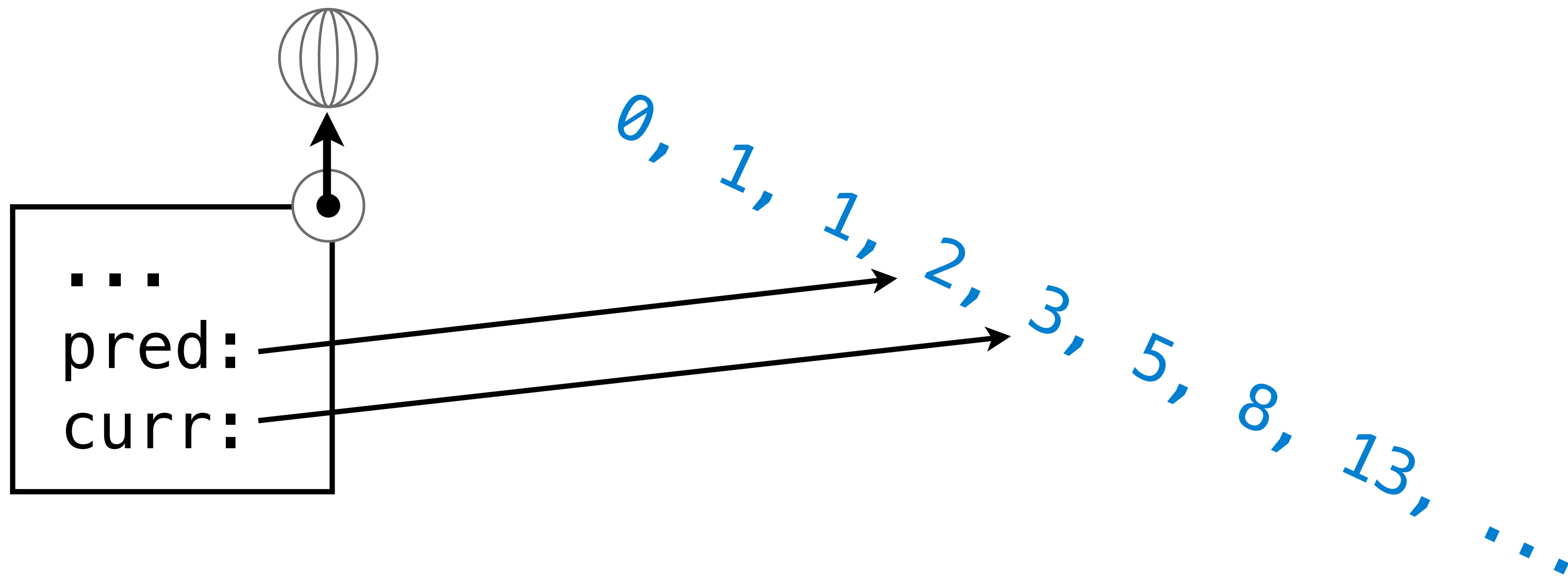

```
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1   # First two Fibonacci numbers
    k = 2               # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

The Fibonacci Sequence



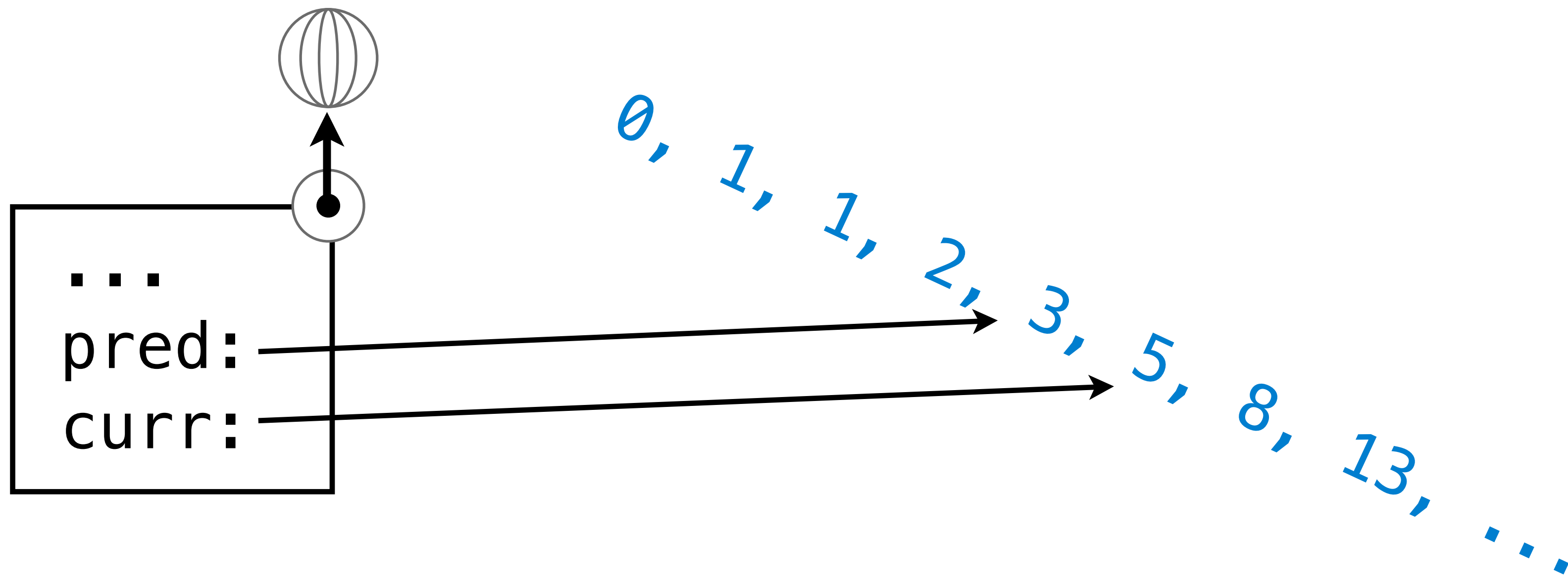

```
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    pred, curr = 0, 1   # First two Fibonacci numbers
    k = 2               # Tracks which Fib number is curr
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

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
```

Project 1: Pig

(Demo)