Lecture 4

Defining Functions

Academic Integrity Quiz

- **Remember**: quiz about the course AI policy
  - Have posted grades for completed quizes
  - Right now, missing ~125 enrolled students
  - If did not receive perfect, take it again
- If you are not aware of the quiz
  - Go to http://www.cs.cornell.edu/courses/cs1110/
  - Click **Academic Integrity** in side bar
  - Read and take quiz in CMS

Recall: Modules

- Modules provide extra functions, variables
  - **Example**: math provides math.cos(), math.pi
  - Access them with the `import` command
- Python provides a lot of them for us
- **This Lecture**: How to make modules
  - Komodo Edit to *make* a module
  - Python to *use* the module

Two different programs

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
>>> plus(23)
24
>>>
```

Function Definition

- Defines what function **does**

```
def plus(n):
    return n+1
```

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
>>> plus(23)
24
>>>
```

Function Definition

- Defines what function **does**

```
def plus(n):
        return n+1
```

Function **Header**

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
>>> plus(23)
24
>>>
```

Function Definition

- Defines what function **does**

```
def plus(n):
      return n+1
```

Function **Header**

Function **Body** (indented)

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

We Write Programs to Do Things

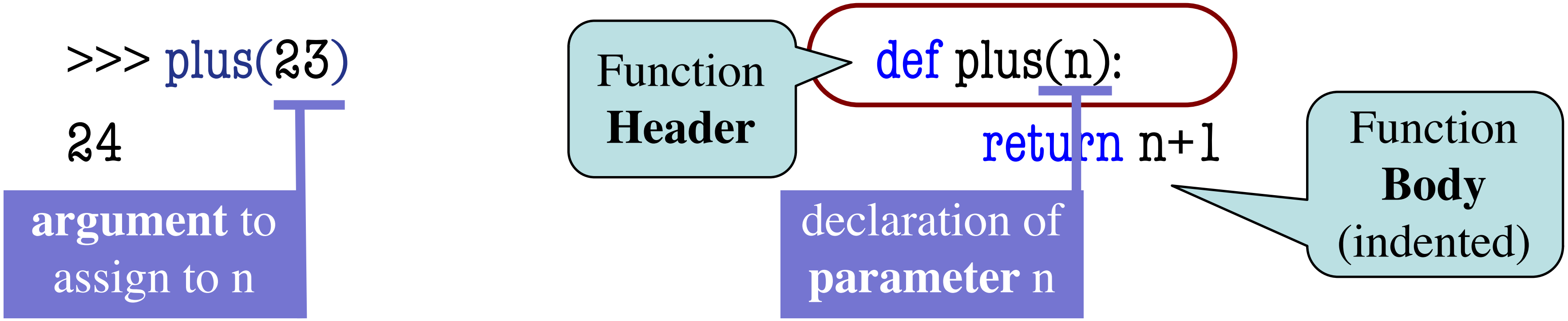
- Functions are the **key doers**

Function Call

- Command to **do** the function

Function Definition

- Defines what function **does**



```
>>> plus(23)
24
```

argument to assign to n

```
def plus(n):
    return n+1
```

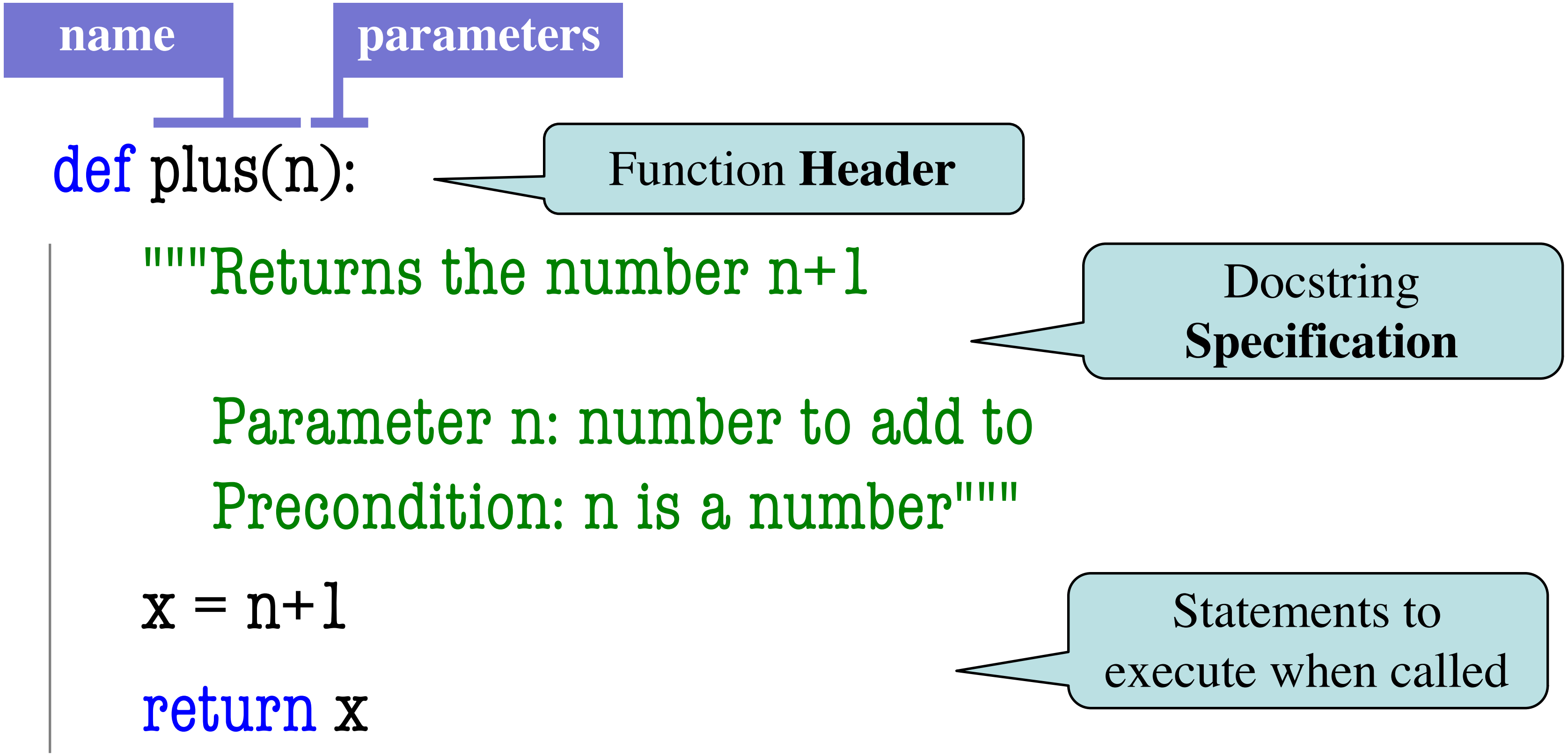
Function **Header**

declaration of **parameter** n

Function **Body** (indented)

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

Anatomy of a Function Definition



```
def plus(n):
    """Returns the number n+1

        Parameter n: number to add to
        Precondition: n is a number"""
    x = n+1
    return x
```

Function **Header**

Docstring **Specification**

Statements to execute when called

Anatomy of a Function Definition

name

parameters

```
def plus(n):
    """Returns the number n+1

       Parameter n: number to add to
       Precondition: n is a number"""
    x = n+1
    return x
```

Function **Header**

Docstring
Specification

Statements to
execute when called

The vertical line
indicates indentation

Use vertical lines when you write Python on **exams** so we can see indentation

The **return** Statement

- **Format**: return <*expression*>
  - Used to evaluate ***function call*** (as an expression)
  - Also stops executing the function!
  - Any statements after a **return** are ignored
- **Example**: temperature converter function

```
def to_centigrade(x):
    """Returns: x converted to centigrade"""
    return 5*(x-32)/9.0
```

A More Complex Example

Function Definition

```
def foo(a,b):
    """Return something
       Param a: number
       Param b: number"""
    x = a
    y = b
    return x*y+y
```

Function Call

```
>>> x = 2
>>> foo(3,4)
```

x ?

What is in the box?

A More Complex Example

Function Definition

```
def foo(a,b):
    """Return something
       Param a: number
       Param b: number"""
    x = a
    y = b
    return x*y+y
```

Function Call

```
>>> x = 2
>>> foo(3,4)
```

x ?

What is in the box?

A: 2
B: 3
C: 16
D: Nothing!
E: I do not know

A More Complex Example

Function Definition

```
def foo(a,b):
    """Return something
       Param a: number
       Param b: number"""
    x = a
    y = b
    return x*y+y
```

Function Call

```
>>> x = 2
>>> foo(3,4)
```

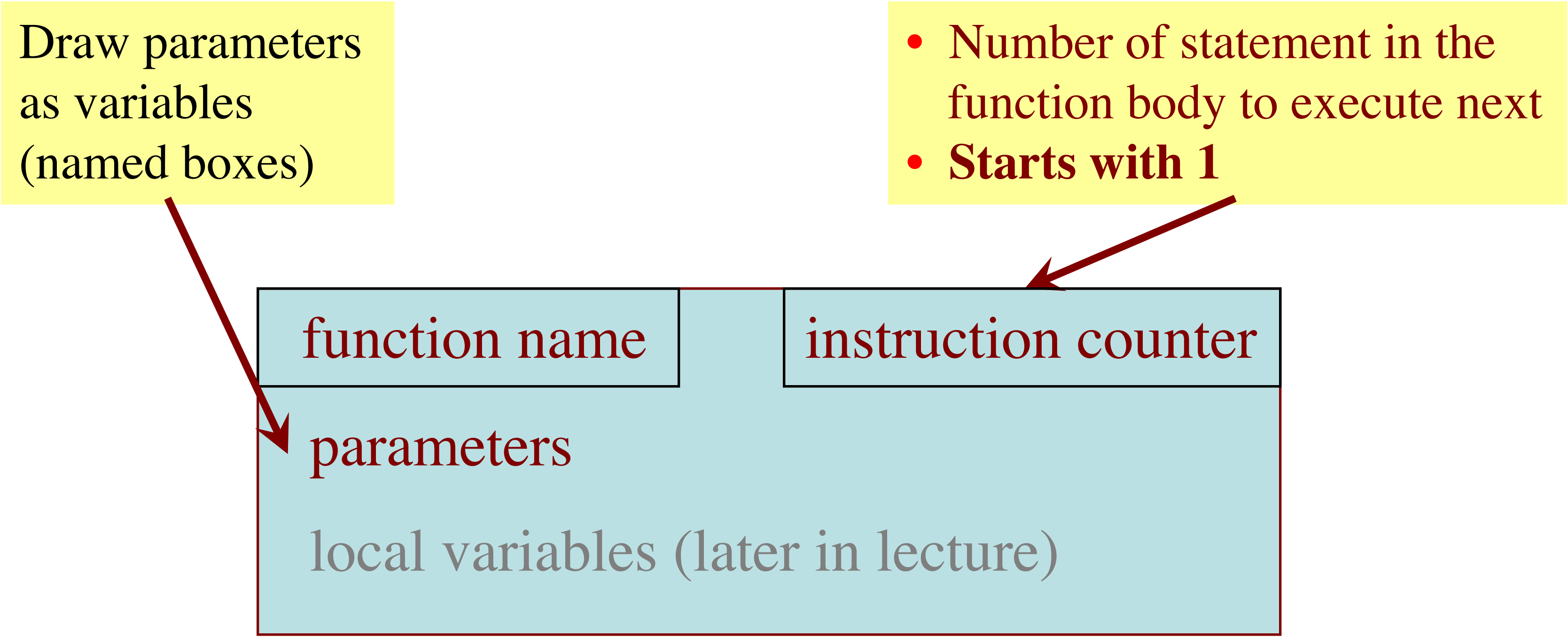
x	?

What is in the box?

A: 2 **CORRECT**
B: 3
C: 16
D: Nothing!
E: I do not know

Understanding How Functions Work

- **Function Frame**: Representation of function call
- A **conceptual model** of Python

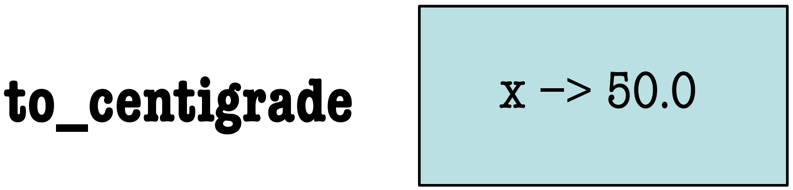


Draw parameters as variables (named boxes)

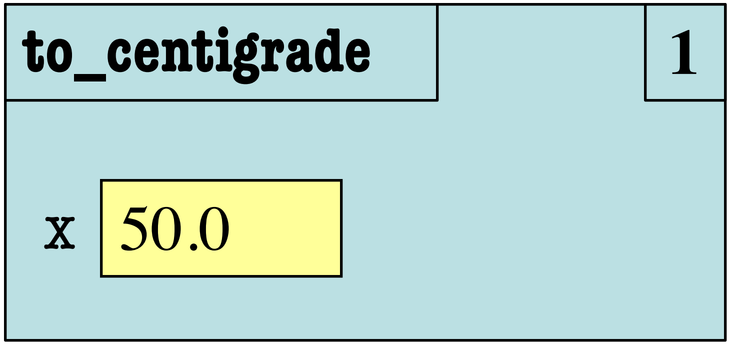
- Number of statement in the function body to execute next
- **Starts with 1**

Text (Section 3.10) vs. Class

Textbook



This Class



Definition:

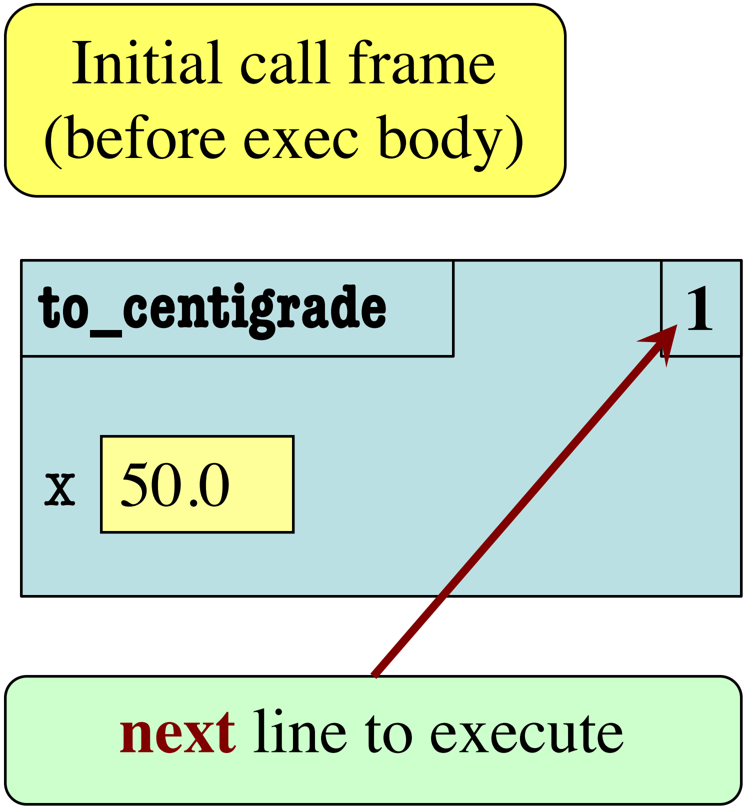
```
def to_centigrade(x):
    return 5*(x-32)/9.0
```

Call: to_centigrade(50.0)

Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```
  def to_centigrade(x):
1   return 5*(x-32)/9.0
```



Initial call frame (before exec body)

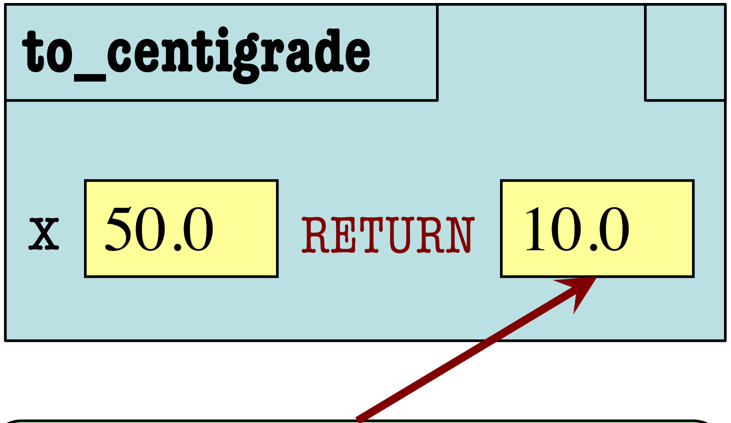
next line to execute

Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```
def to_centigrade(x):
    return 5*(x-32)/9.0
```

Executing the return statement



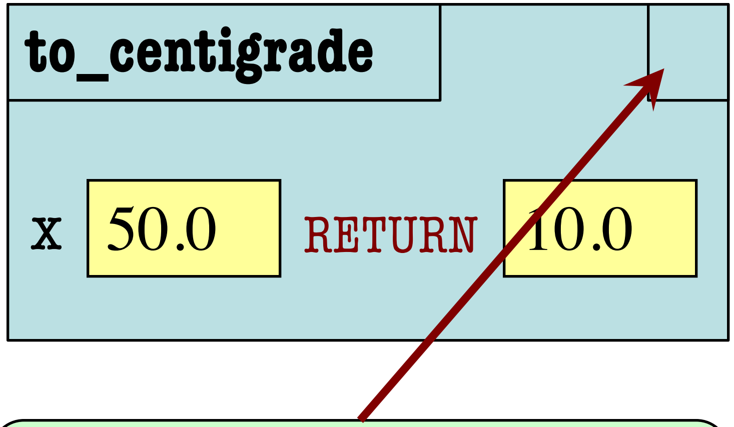
Return statement creates a special variable for result

Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```
def to_centigrade(x):
    return 5*(x-32)/9.0
```

Executing the return statement



The return terminates; no next line to execute

Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```
def to_centigrade(x):
    return 5*(x-32)/9.0
```

ERASE WHOLE FRAME

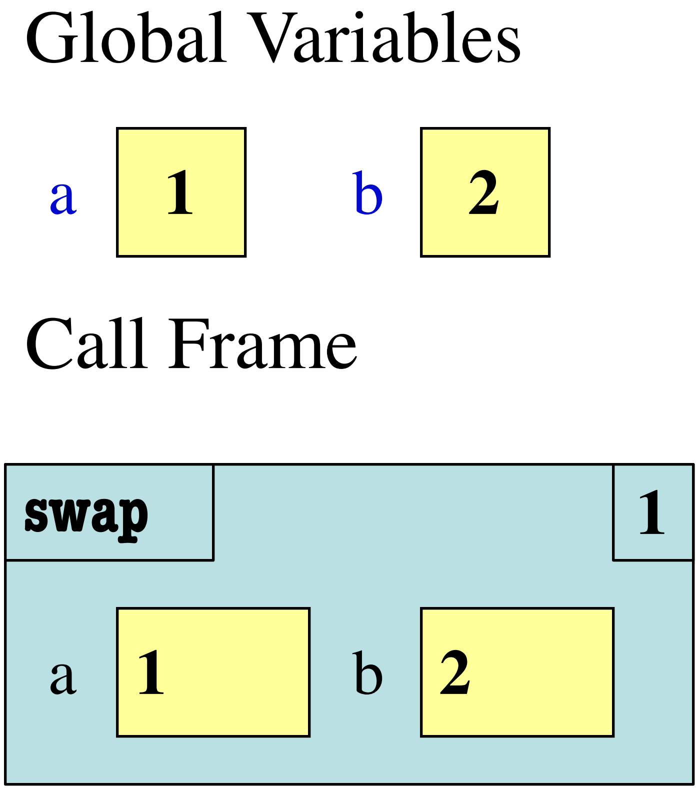
But don’t actually erase on an exam

Call Frames vs. Global Variables

The specification is a **lie**:

```
  def swap(a,b):
      """Swap global a & b"""
1     tmp = a
2     a = b
3     b = tmp
```

```
>>> a = 1
>>> b = 2
>>> swap(a,b)
```



Global Variables

a 1 b 2

Call Frame

swap 1

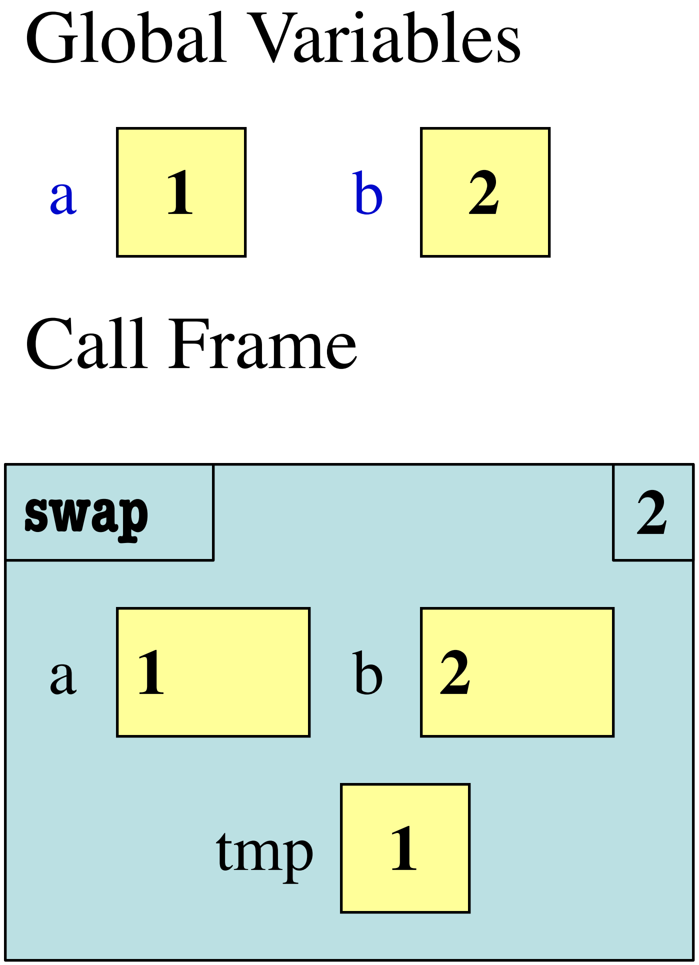
a 1 b 2

Call Frames vs. Global Variables

The specification is a **lie**:

```
    def swap(a,b):
        """Swap global a & b"""
1       tmp = a
2       a = b
3       b = tmp
```

```
>>> a = 1
>>> b = 2
>>> swap(a,b)
```

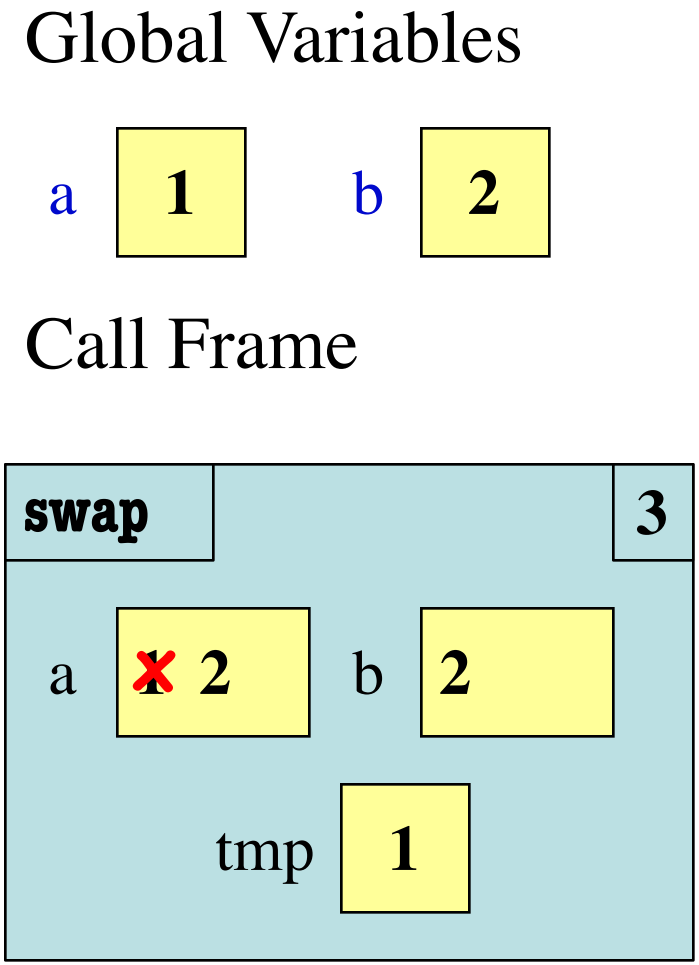


Call Frames vs. Global Variables

The specification is a **lie**:

```
  def swap(a,b):
      """Swap global a & b"""
1     tmp = a
2     a = b
3     b = tmp
```

```
>>> a = 1
>>> b = 2
>>> swap(a,b)
```

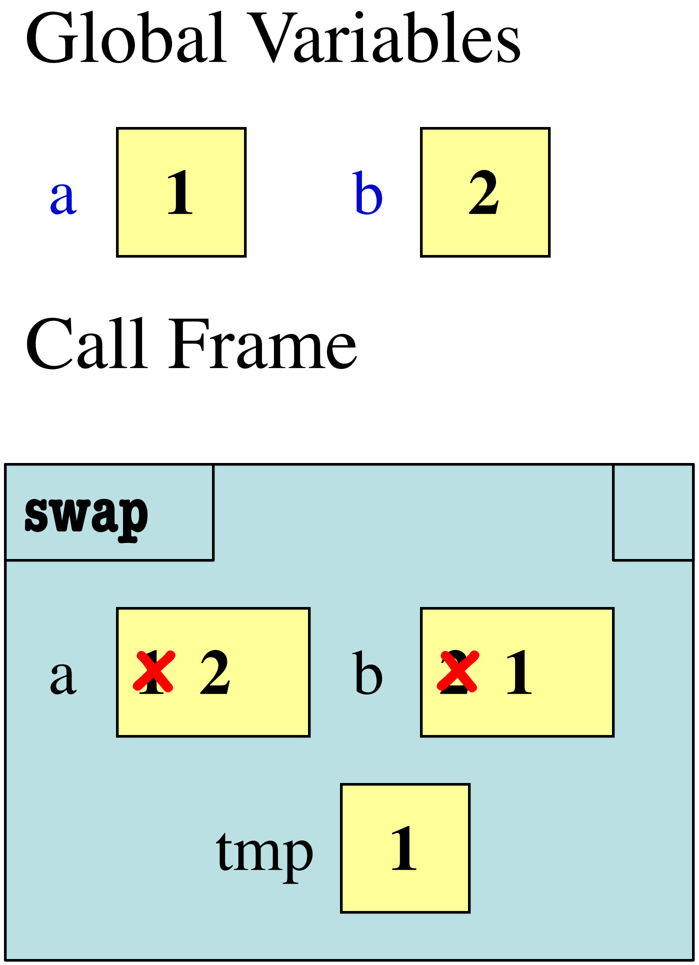


Call Frames vs. Global Variables

The specification is a **lie**:

```
def swap(a,b):
    """Swap global a & b"""
    tmp = a
    a = b
    b = tmp
```

```
>>> a = 1
>>> b = 2
>>> swap(a,b)
```



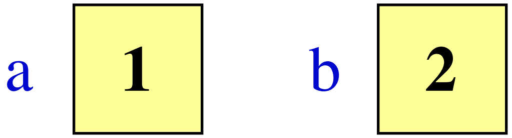
Call Frames vs. Global Variables

The specification is a **lie**:

```
def swap(a,b):
    """Swap global a & b"""
    tmp = a
    a = b
    b = tmp
```

```
>>> a = 1
>>> b = 2
>>> swap(a,b)
```

Global Variables

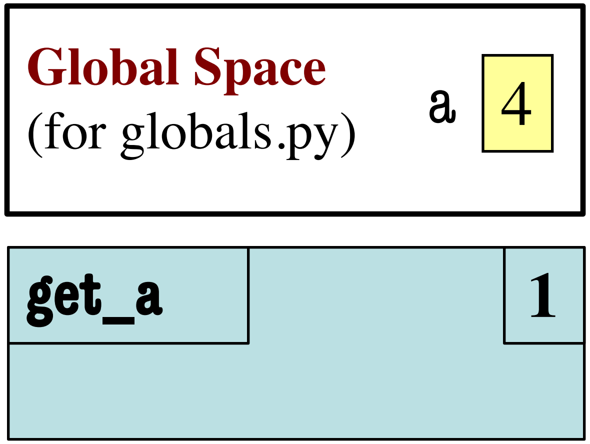


Call Frame

ERASE THE FRAME

Function Access to Global Space

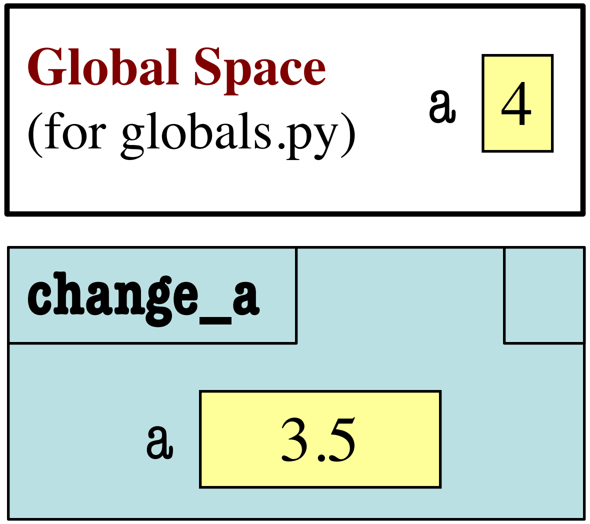
- All function definitions are in some module
- Call can access global space for **that module**
  - math.cos: global for math
  - temperature.to_centigrade uses global for temperature
- But **cannot** change values
  - Assignment to a global makes a new local variable!
  - Why we limit to constants



```
# globals.py
"""Show how globals work"""
a = 4 # global space


def get_a():
    return a # returns global
```

Function Access to Global Space

- All function definitions are in some module
- Call can access global space for **that module**
  - math.cos: global for math
  - temperature.to_centigrade uses global for temperature
- But **cannot** change values
  - Assignment to a global makes a new local variable!
  - Why we limit to constants



```
# globals.py
"""Show how globals work"""
a = 4 # global space
def change_a():
    a = 3.5 # local variable
    return a
```

Exercise Time

Function Definition

```
def foo(a,b):
    """Return something
       Param x: a number
       Param y: a number"""
    x = a
    y = b
    return x*y+y
```

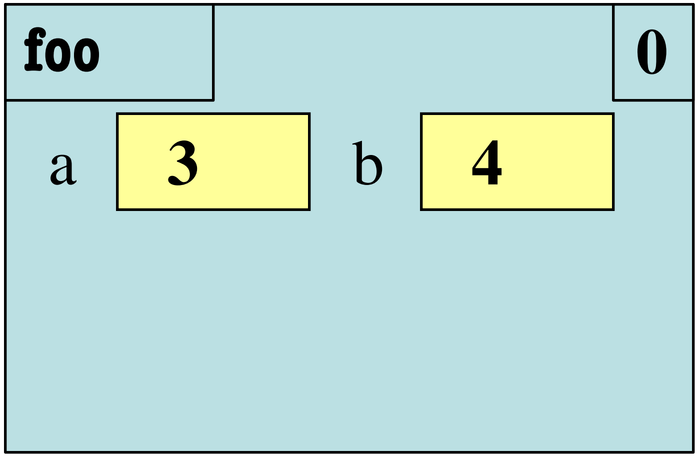
Function Call

```
>>> x = foo(3,4)
```

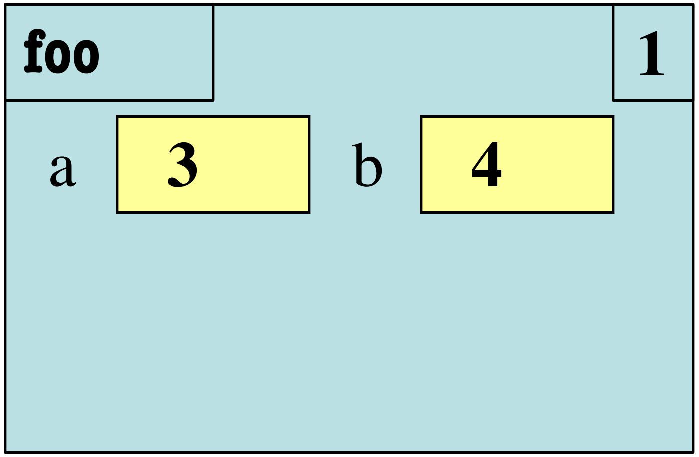
What does the frame look like at the **start**?

Which One is Closest to Your Answer?

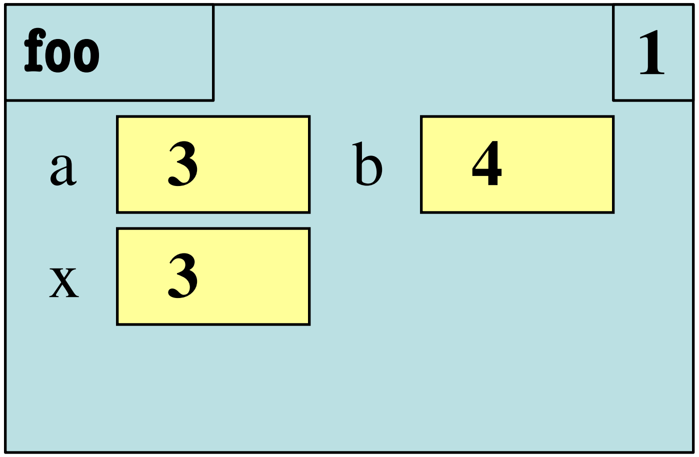
A:



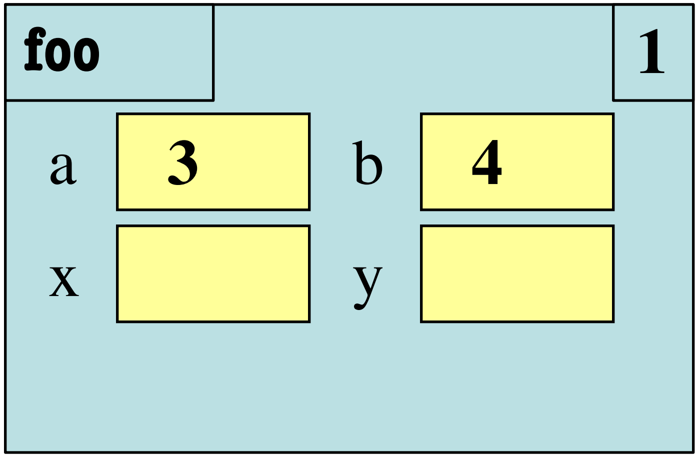
B:



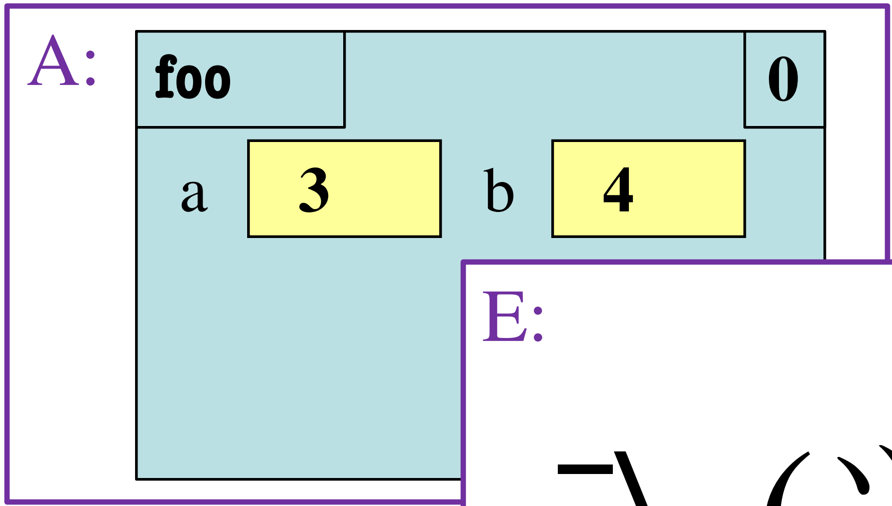
C:

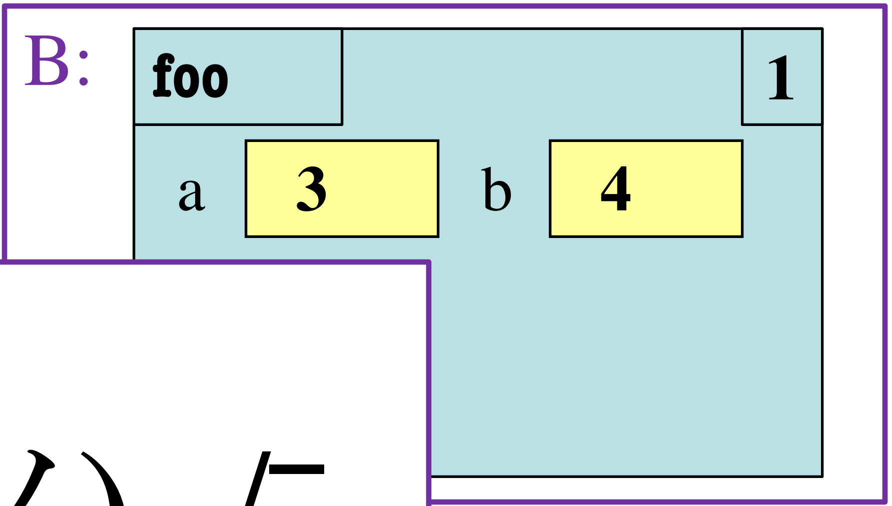


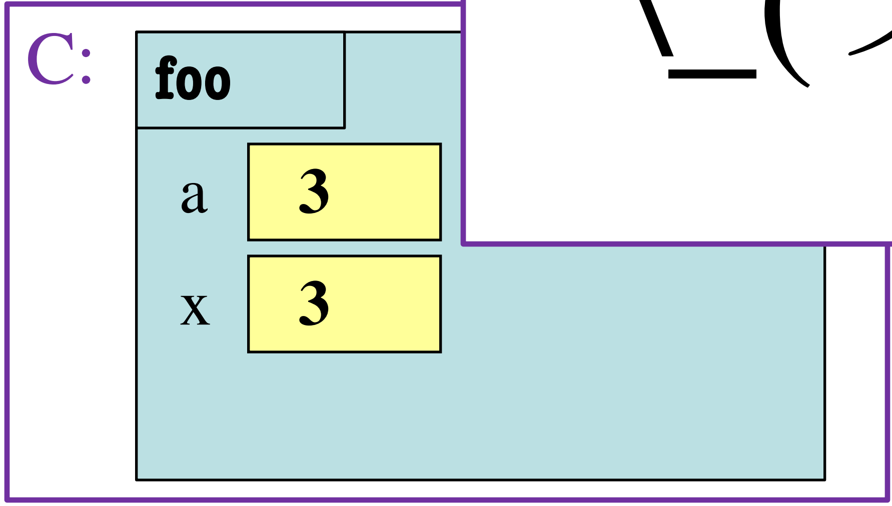
D:

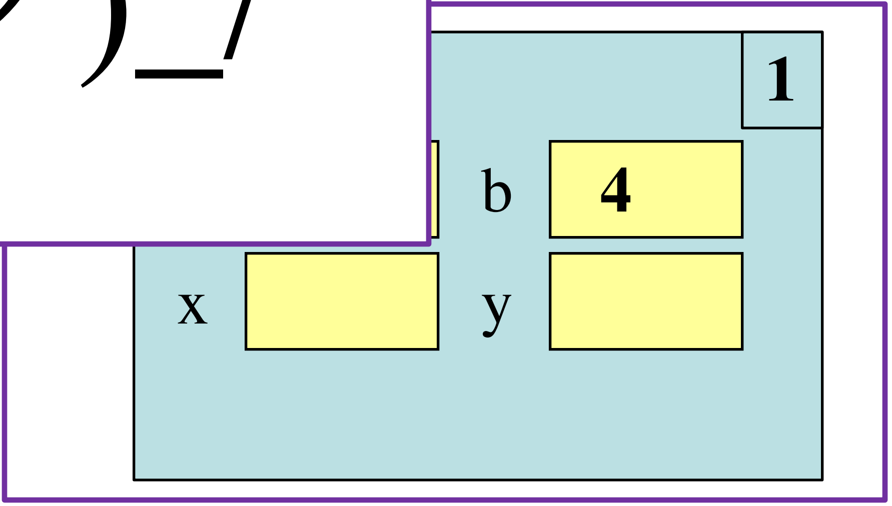


Which One is Closest to Your Answer?









E:

¯_(ツ)_/¯

Exercise Time

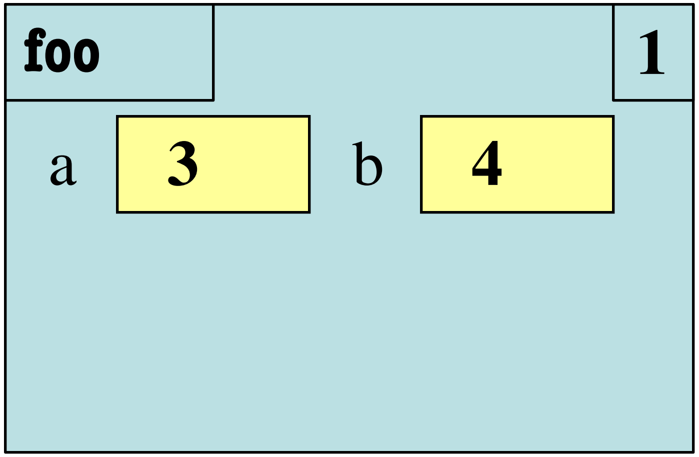
Function Definition

```
  def foo(a,b):
      """Return something
         Param x: a number
         Param y: a number"""
1     x = a
2     y = b
3     return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

B:



foo 1

a 3 b 4

Exercise Time

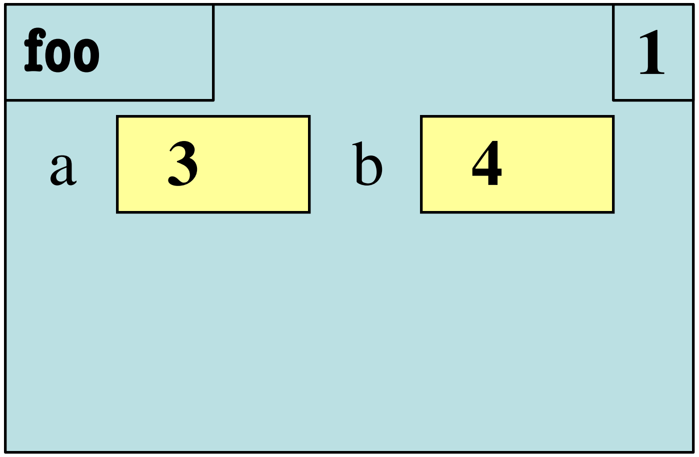
Function Definition

```
def foo(a,b):
    """Return something
       Param x: a number
       Param y: a number"""
1   x = a
2   y = b
3   return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

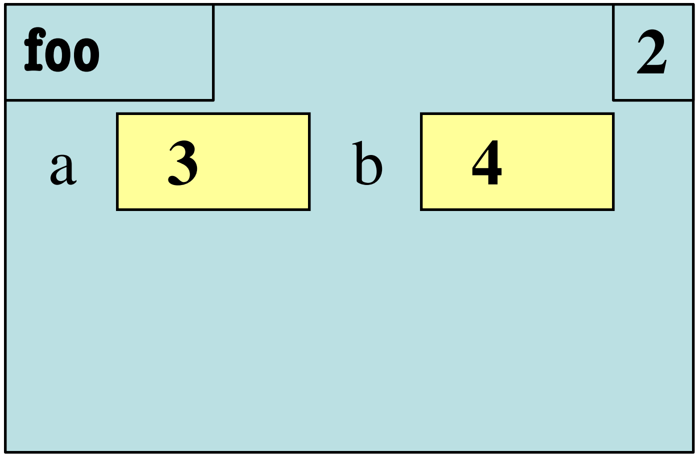
B:



What is the **next step**?

Which One is Closest to Your Answer?

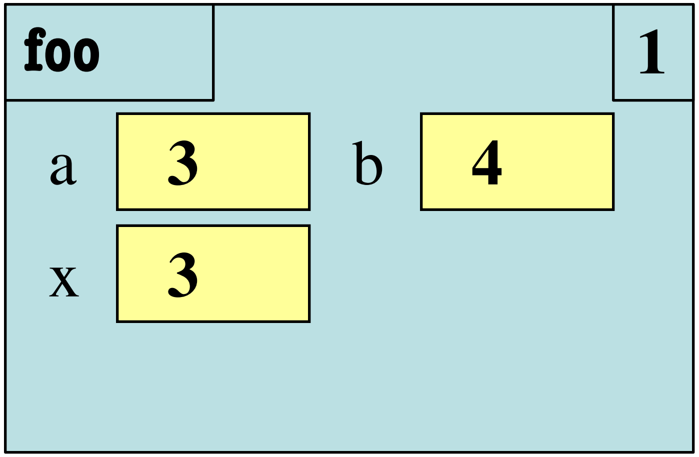
A:



foo 2

a 3 b 4

B:

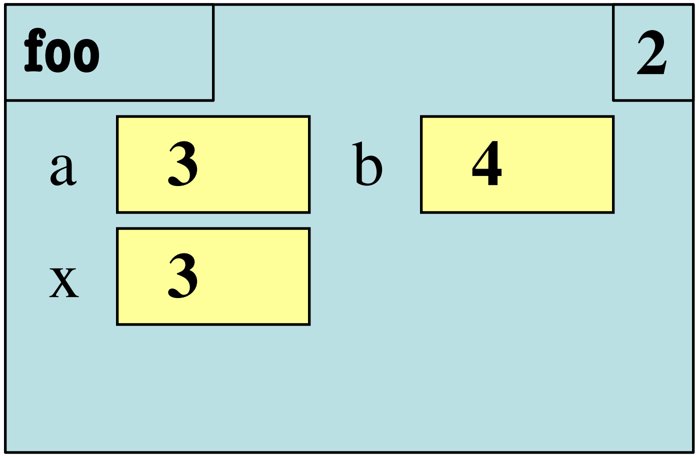


foo 1

a 3 b 4

x 3

C:

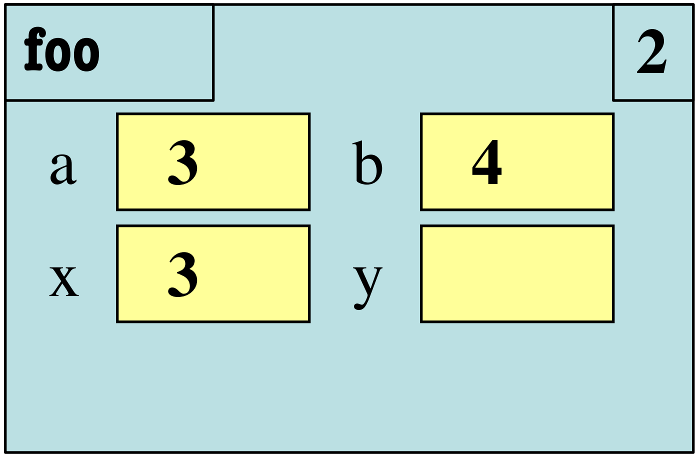


foo 2

a 3 b 4

x 3

D:



foo 2

a 3 b 4

x 3 y

Exercise Time

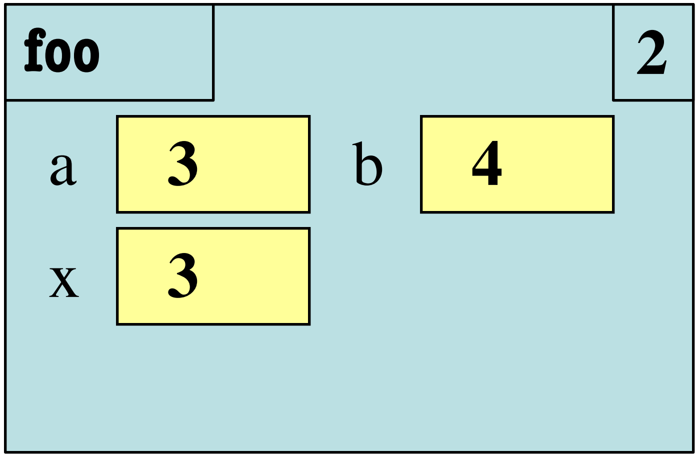
Function Definition

```
def foo(a,b):
    """Return something
       Param x: a number
       Param y: a number"""
1   x = a
2   y = b
3   return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

C:



Exercise Time

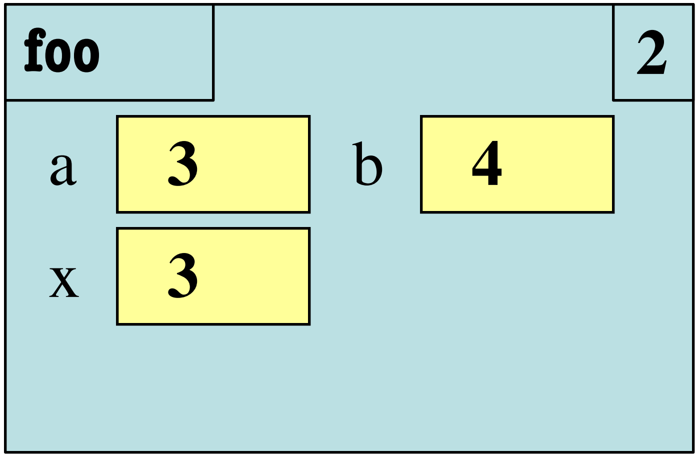
Function Definition

```
  def foo(a,b):
      """Return something
         Param x: a number
         Param y: a number"""
1     x = a
2     y = b
3     return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

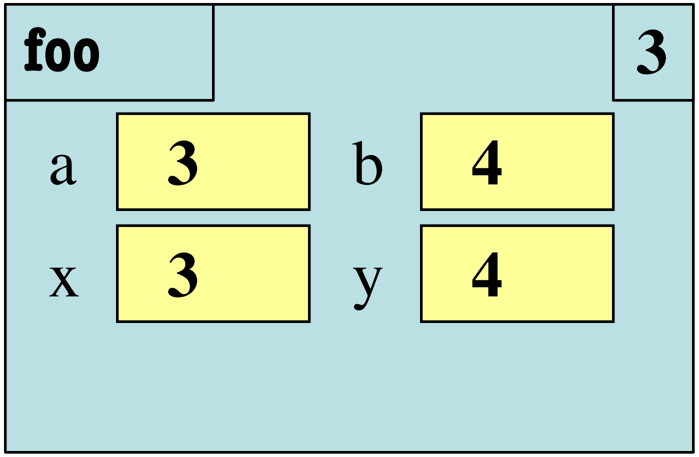
C:



What is the **next step**?

Which One is Closest to Your Answer?

A:

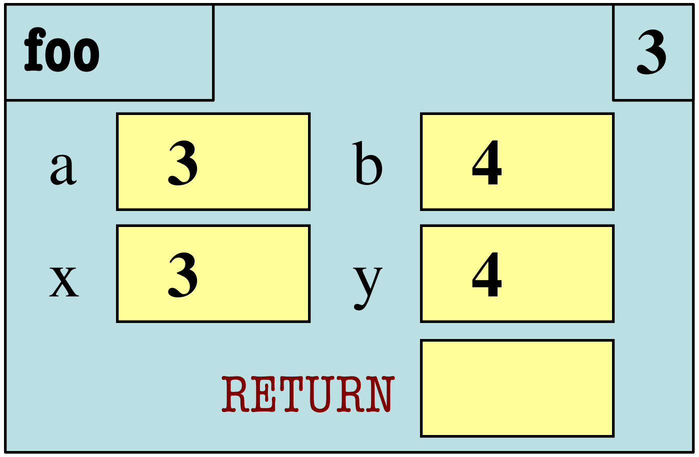


foo 3

a 3 b 4

x 3 y 4

B:



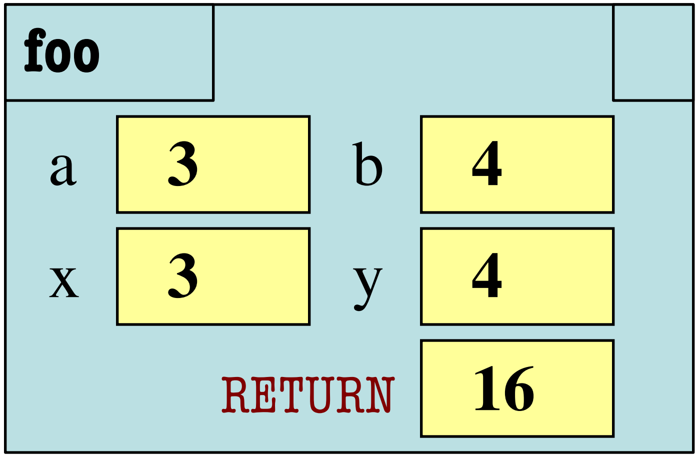
foo 3

a 3 b 4

x 3 y 4

RETURN

C:



foo

a 3 b 4

x 3 y 4

RETURN 16

D:

ERASE THE FRAME

Exercise Time

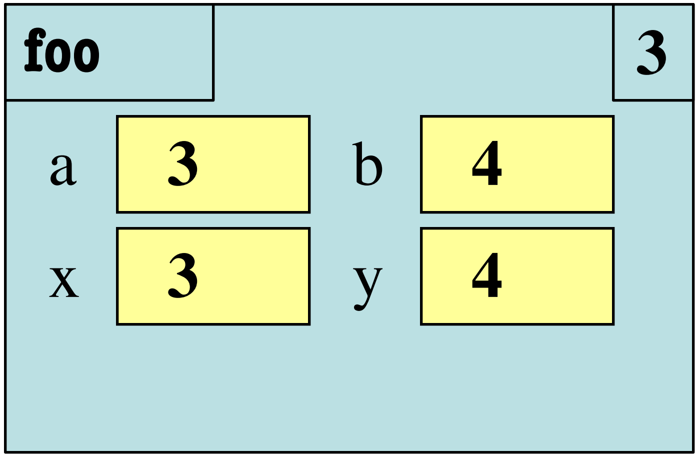
Function Definition

```
def foo(a,b):
    """Return something
       Param x: a number
       Param y: a number"""
1   x = a
2   y = b
3   return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

A:



Exercise Time

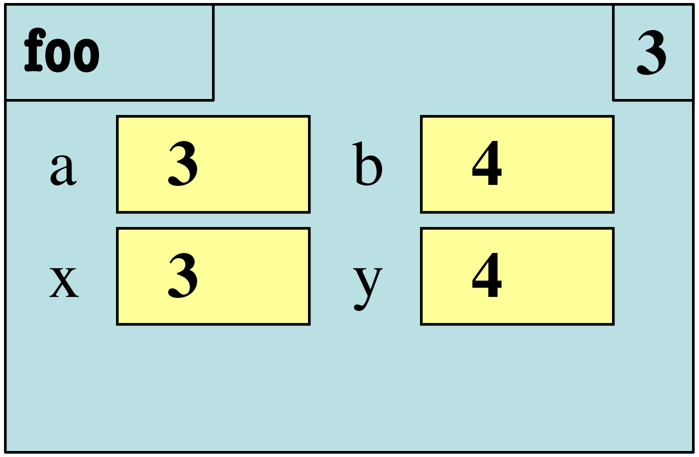
Function Definition

```
def foo(a,b):
    """Return something
       Param x: a number
       Param y: a number"""
1   x = a
2   y = b
3   return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

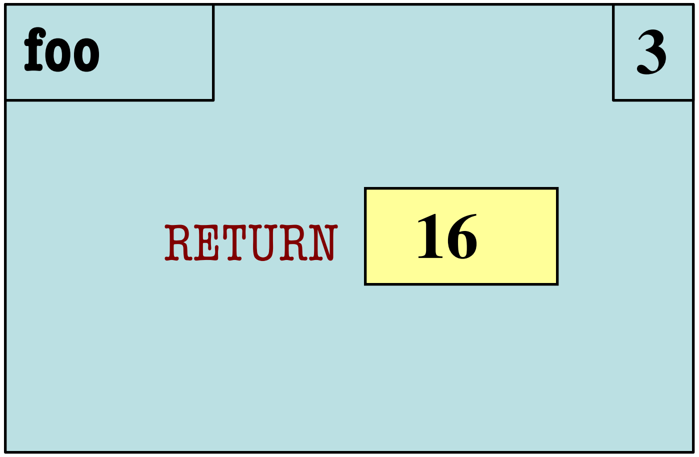
A:



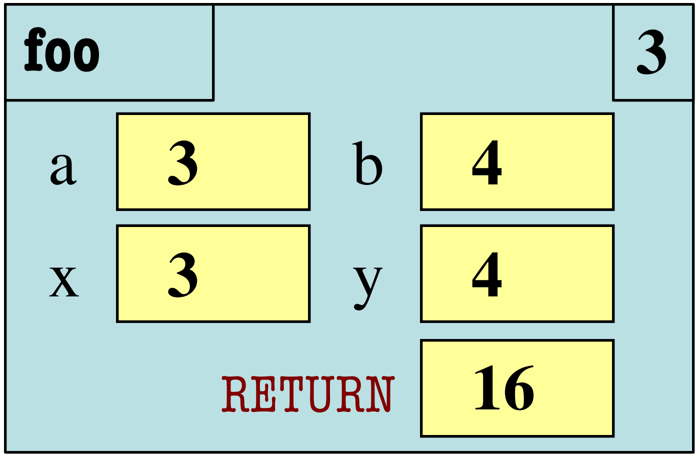
What is the **next step**?

Which One is Closest to Your Answer?

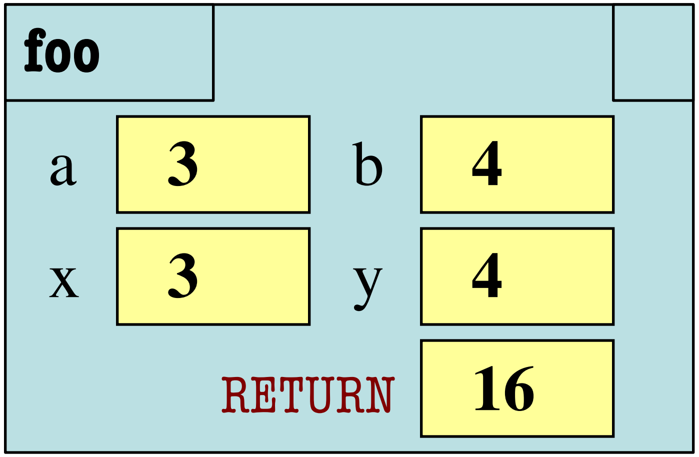
A:



B:



C:



D:

ERASE THE FRAME

Exercise Time

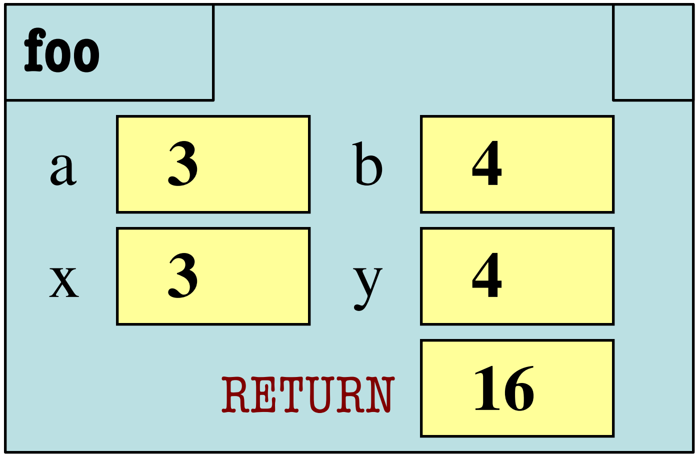
Function Definition

```
def foo(a,b):
    """Return something
       Param x: a number
       Param y: a number"""
    x = a
    y = b
    return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

C:



Exercise Time

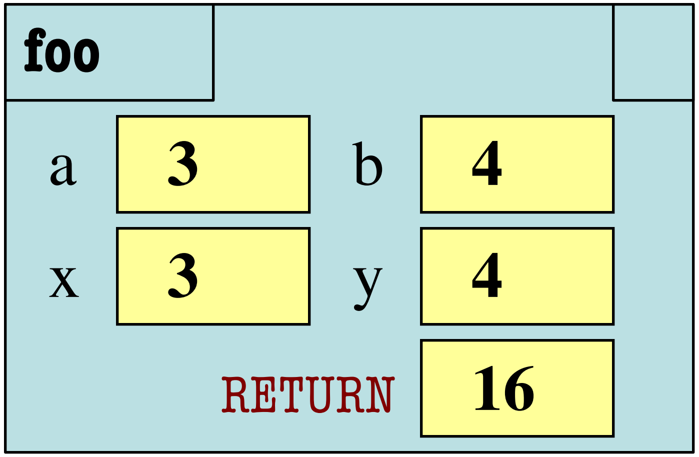
Function Definition

```
def foo(a,b):
    """Return something
       Param x: a number
       Param y: a number"""
    x = a
    y = b
    return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

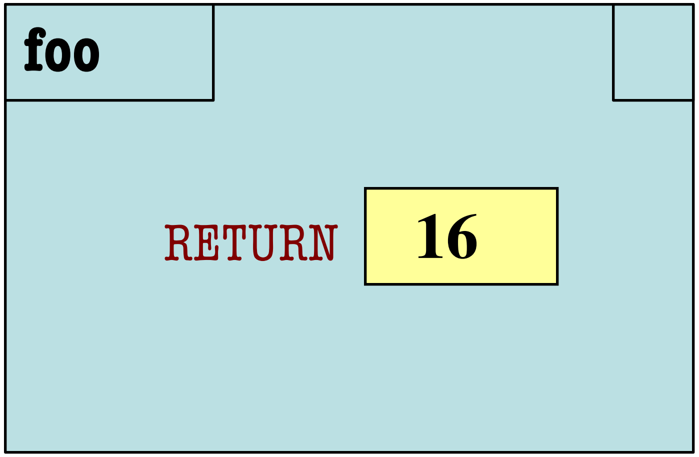
C:



What is the **next step**?

Which One is Closest to Your Answer?

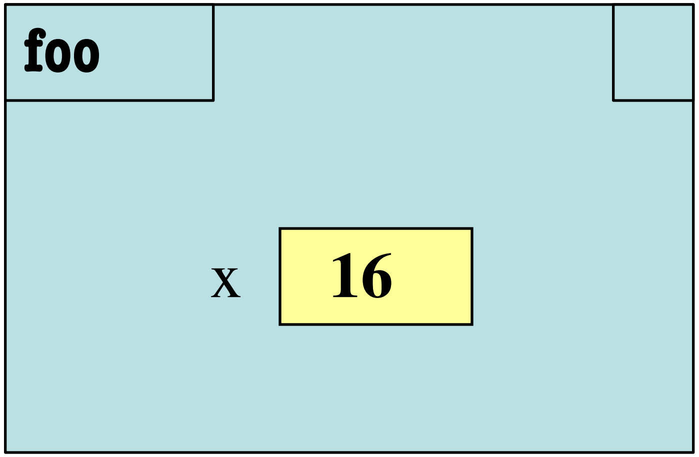
A:



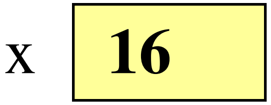
B:

ERASE THE FRAME

C:



D:



ERASE THE FRAME

Exercise Time

Function Definition

```
def foo(a,b):
    """Return something
       Param x: a number
       Param y: a number"""
    x = a
    y = b
    return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

D:

x **16**

ERASE THE FRAME

Exercise Time

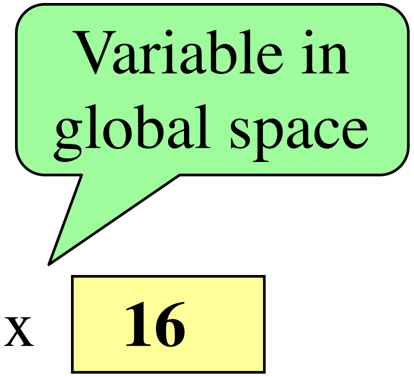
Function Definition

```
def foo(a,b):
    """Return something
       Param x: a number
       Param y: a number"""
    x = a
    y = b
    return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

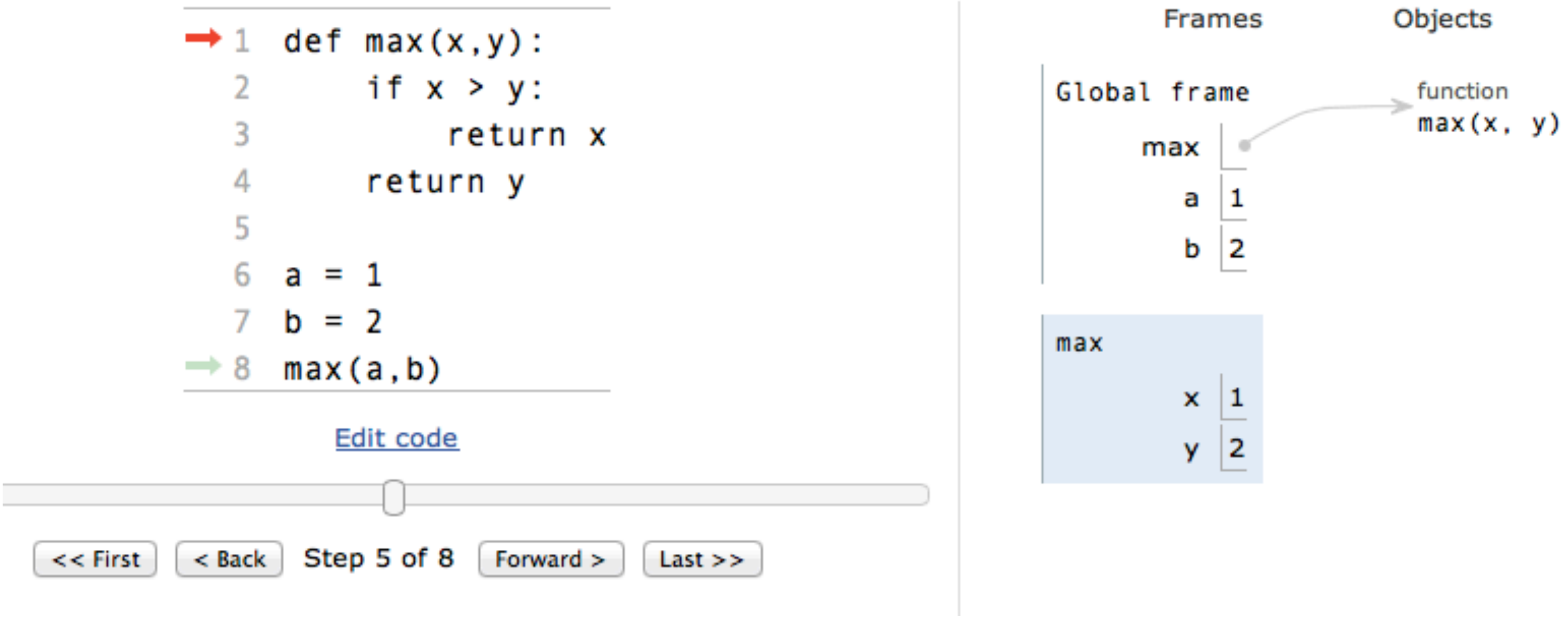
D:



Variable in global space

ERASE THE FRAME

Visualizing Frames: The Python Tutor



```
def max(x,y):
    if x > y:
        return x
    return y

a = 1
b = 2
max(a,b)
```

Step 5 of 8

Frames

Global frame

max

a 1

b 2

max

x 1

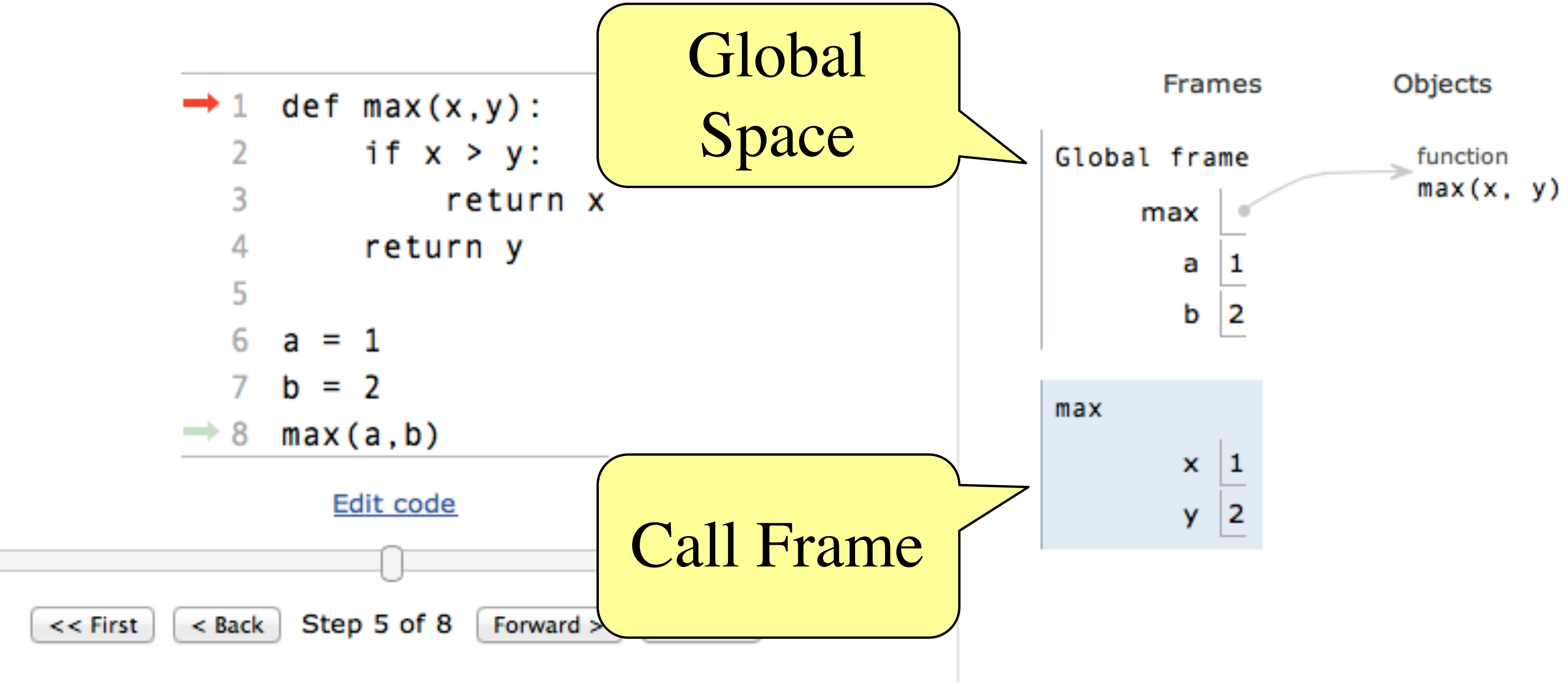
y 2

Objects

function

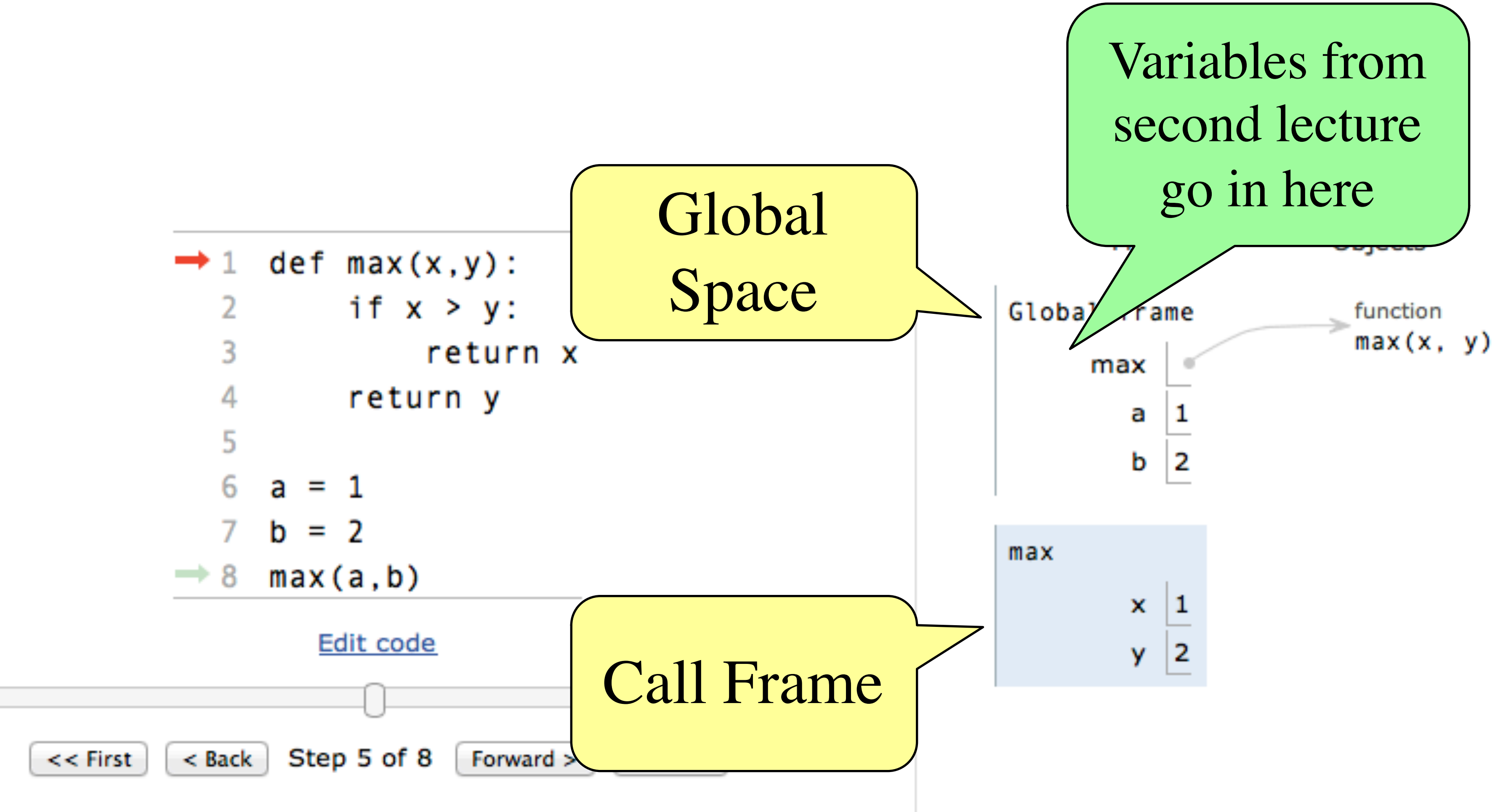
max(x, y)

Visualizing Frames: The Python Tutor



```
def max(x,y):
    if x > y:
        return x
    return y

a = 1
b = 2
max(a,b)
```

Visualizing Frames: The Python Tutor



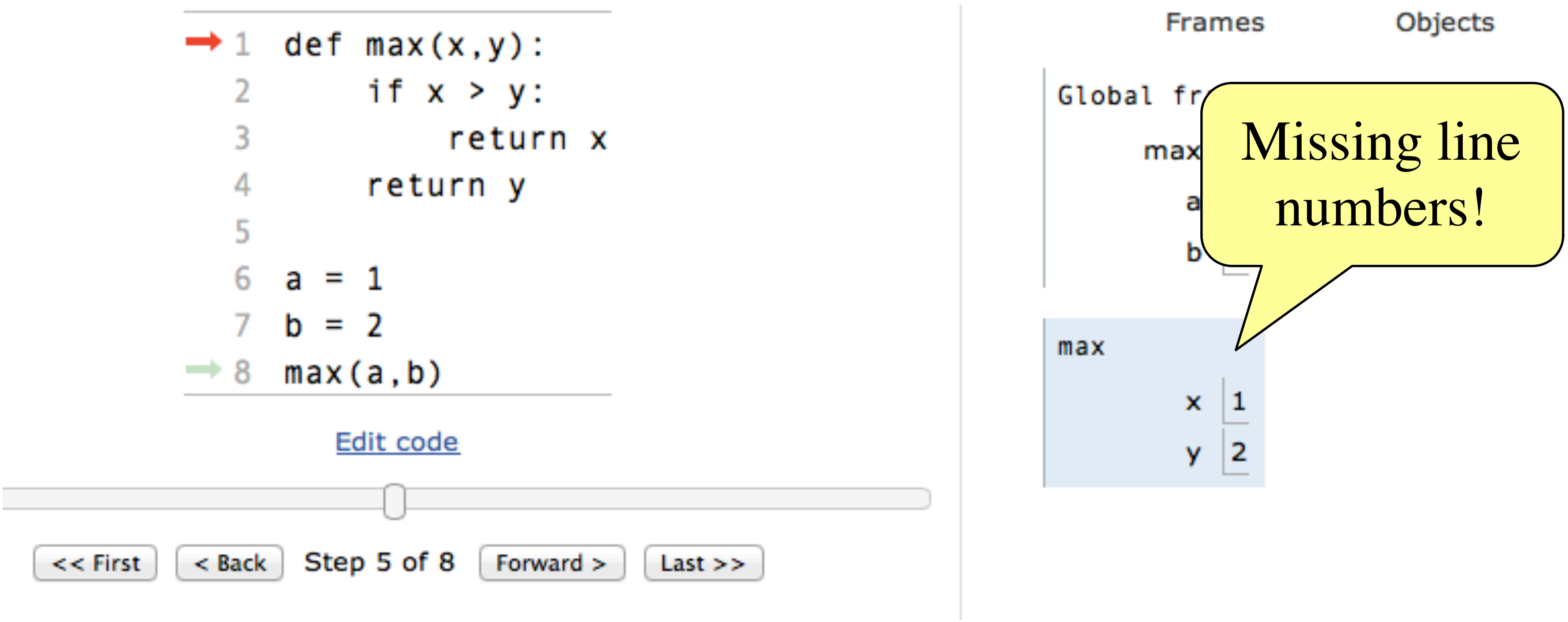
```
def max(x,y):
    if x > y:
        return x
    return y

a = 1
b = 2
max(a,b)
```

Global Space

Variables from second lecture go in here

Call Frame

Visualizing Frames: The Python Tutor

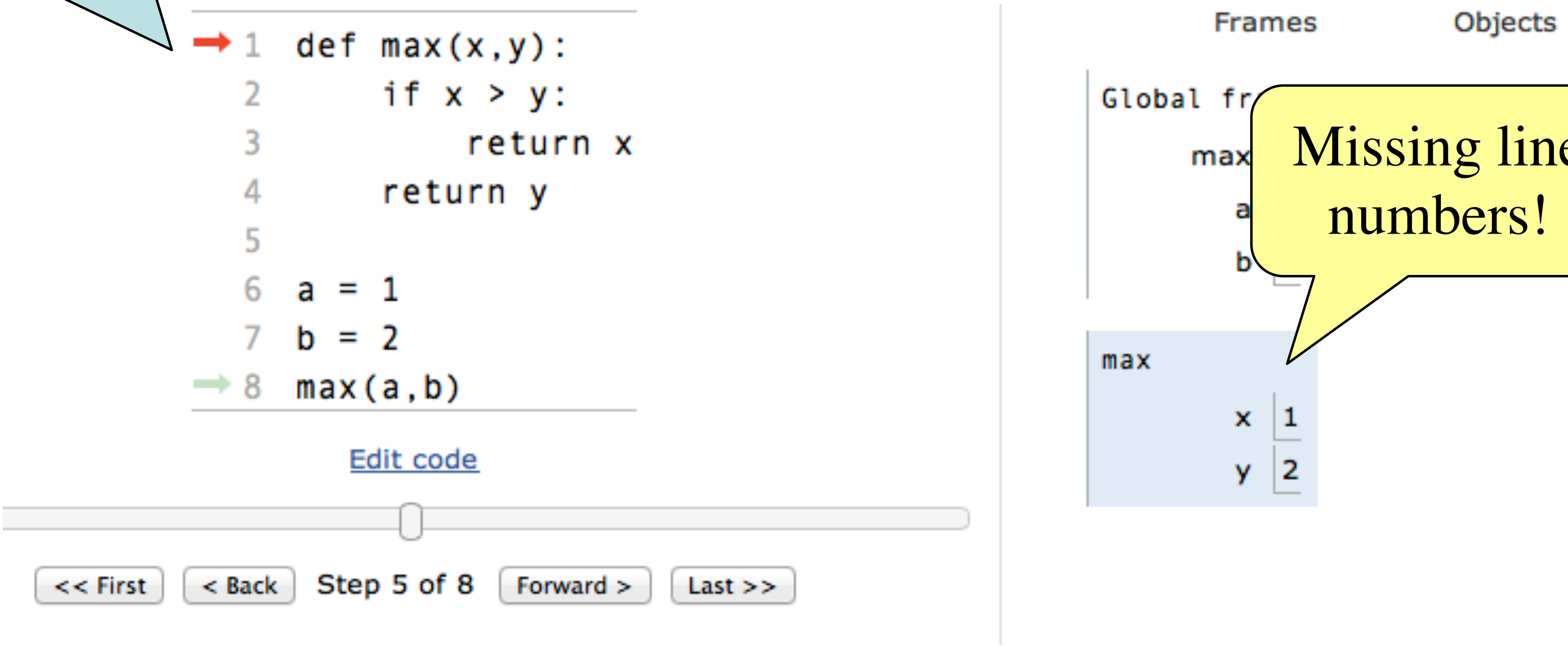


```
def max(x,y):
    if x > y:
        return x
    return y

a = 1
b = 2
max(a,b)
```

Missing line numbers!

Visualizing Frames: The Python Tutor

Line number marked here (sort-of)



```
→ 1  def max(x,y):
  2      if x > y:
  3          return x
  4      return y
  5
  6  a = 1
  7  b = 2
→ 8  max(a,b)
```

Missing line numbers!

Next Time: Concrete Examples