

Lecture 6

Visualizing Functions

Announcements for this Lecture

Last Call

- Quiz: About the Course
- Take it by tomorrow
- Also remember survey



Assignment 1

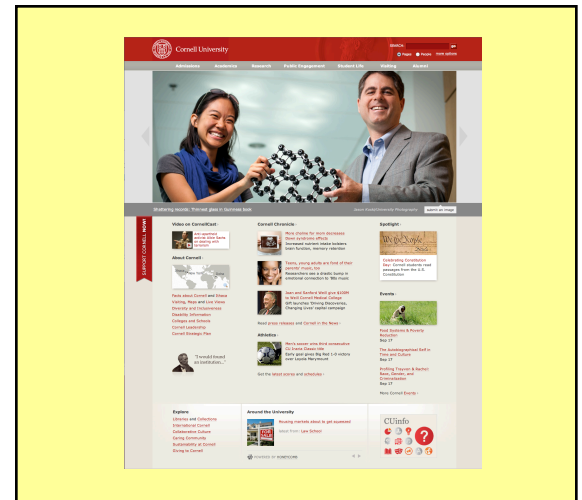
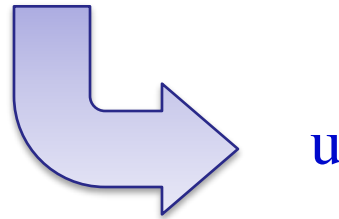
- Assignment 1 is live
 - Posted on web page
 - Due Thur, Sep. 17th
 - Due in place of Lab 4
- Lab 3 will help a lot
 - Testing is a major part
 - Try to finish it first
 - But start this Saturday!

One-on-One Sessions

- Still ongoing: 1/2-hour one-on-one sessions
 - To help prepare you for the assignment
 - **Primarily for students with little experience**
- There are still some spots available
 - Sign up for a slot in CMS
- Will keep running after **September 17**
 - Will open additional slots after the due date
 - Will help students revise Assignment 1

A1: The Module urllib2

- Module urllib2 is used to read web pages
 - Function urlopen creates a url object
 - `u = urllib2.urlopen('http://www.cornell.edu')`



- url has a method called read()
 - Returns contents of web page
 - **Usage:** `s = u.read()` # s is a string

A Motivating Example


Function Definition

```
def foo(a,b):  
    """Do 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 Motivating Example


Function Definition

```
def foo(a,b):  
    """Do 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 Motivating Example

Function Definition

```
def foo(a,b):  
    """Do 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

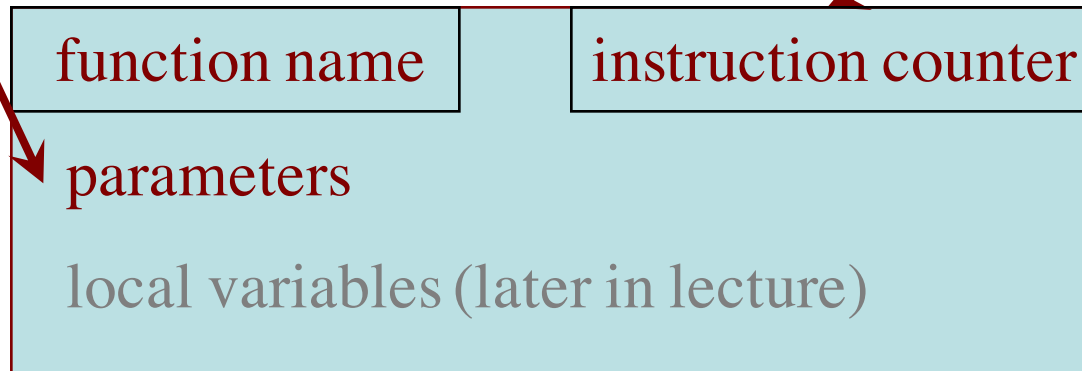
How Do Functions Work?

Draw template on a piece of paper

- **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

to_centigrade

$x \rightarrow 50.0$

This Class

to_centigrade

1

x 50.0

Definition:

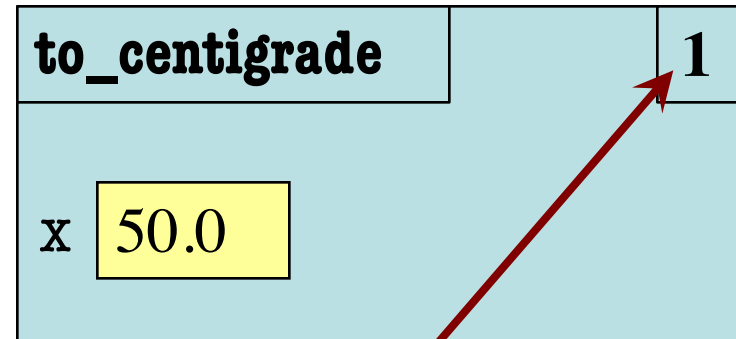
```
def to_centigrade(x):  
    | return 5*(x-32)/9.0
```

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

Initial call frame
(before exec body)



next line to execute

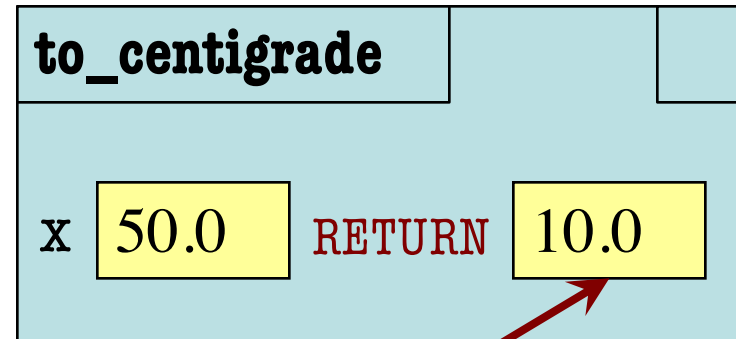
```
def to_centigrade(x):  
1 | return 5*(x-32)/9.0
```

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
```

Executing the
return statement



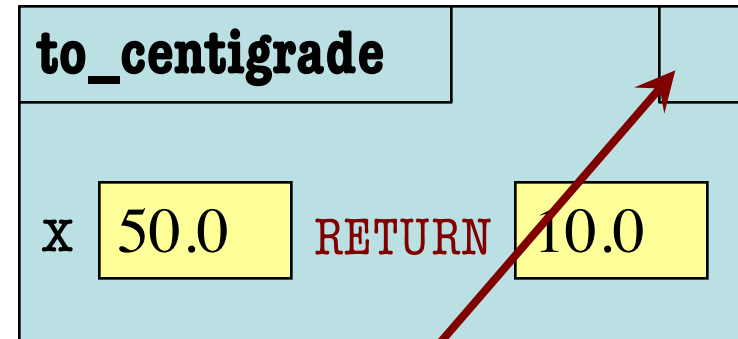
Return statement creates a
special variable for result

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):  
1 | 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

ERASE WHOLE FRAME

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

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
```

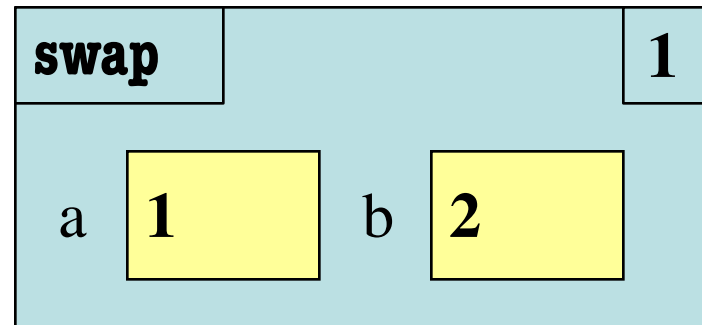
```
>>> b = 2
```

```
>>> swap(a,b)
```

Global Variables

a **1** b **2**

Call Frame



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
```

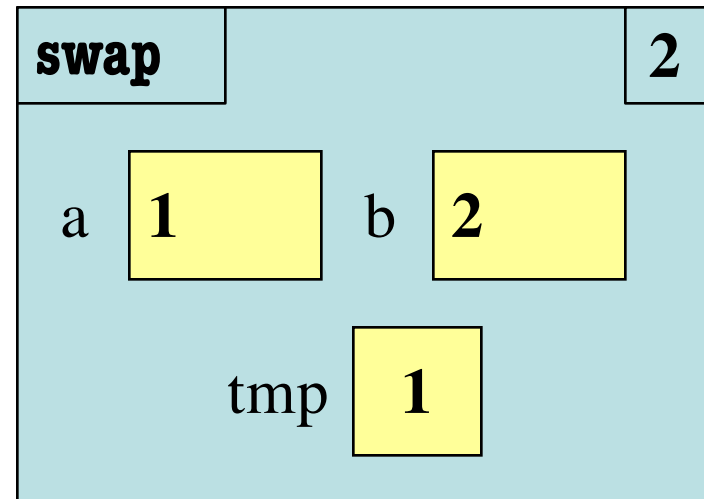
```
>>> b = 2
```

```
>>> swap(a,b)
```

Global Variables

a **1** b **2**

Call Frame



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
```

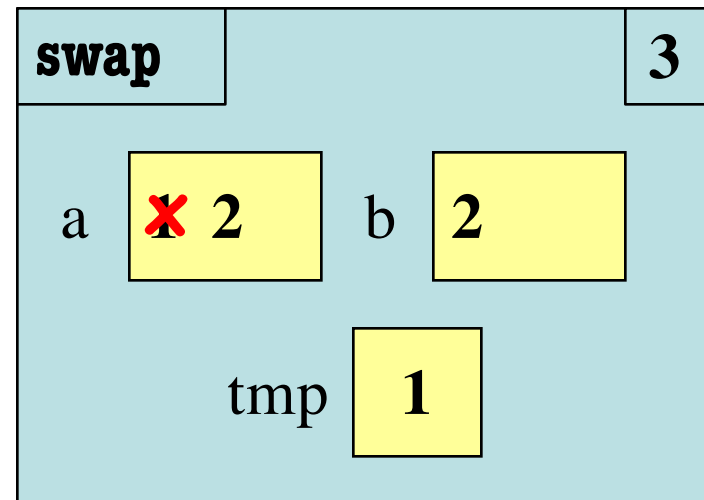
```
>>> b = 2
```

```
>>> swap(a,b)
```

Global Variables

a **1** b **2**

Call Frame



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
```

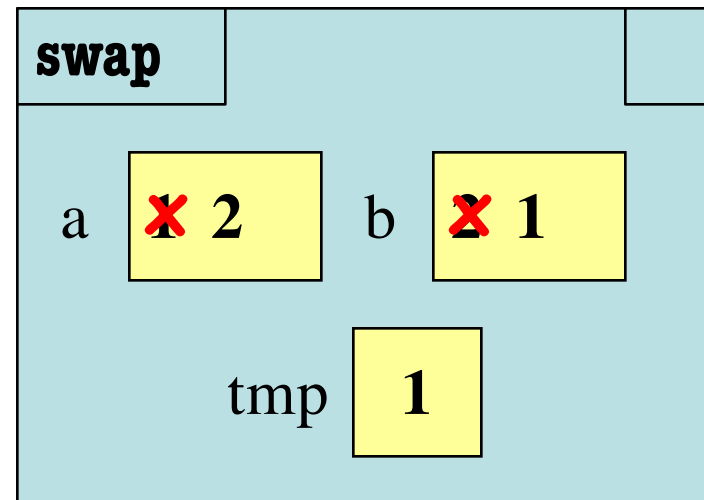
```
>>> b = 2
```

```
>>> swap(a,b)
```

Global Variables

a **1** b **2**

Call Frame



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

ERASE THE FRAME

Visualizing Frames: The Python Tutor

```
→ 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)
```

[Edit code](#)

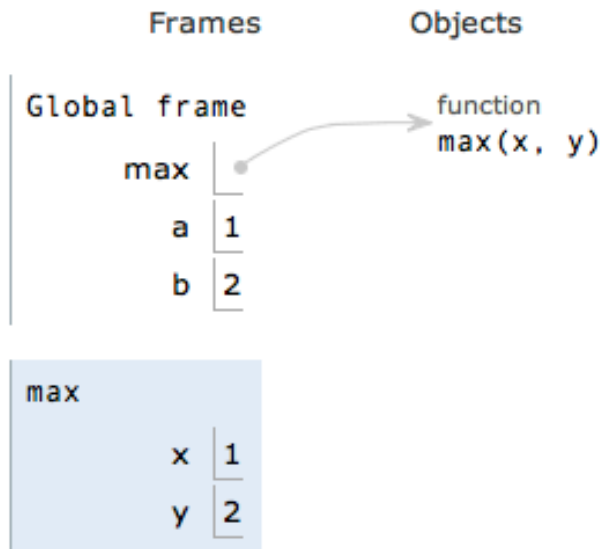
<< First

< Back

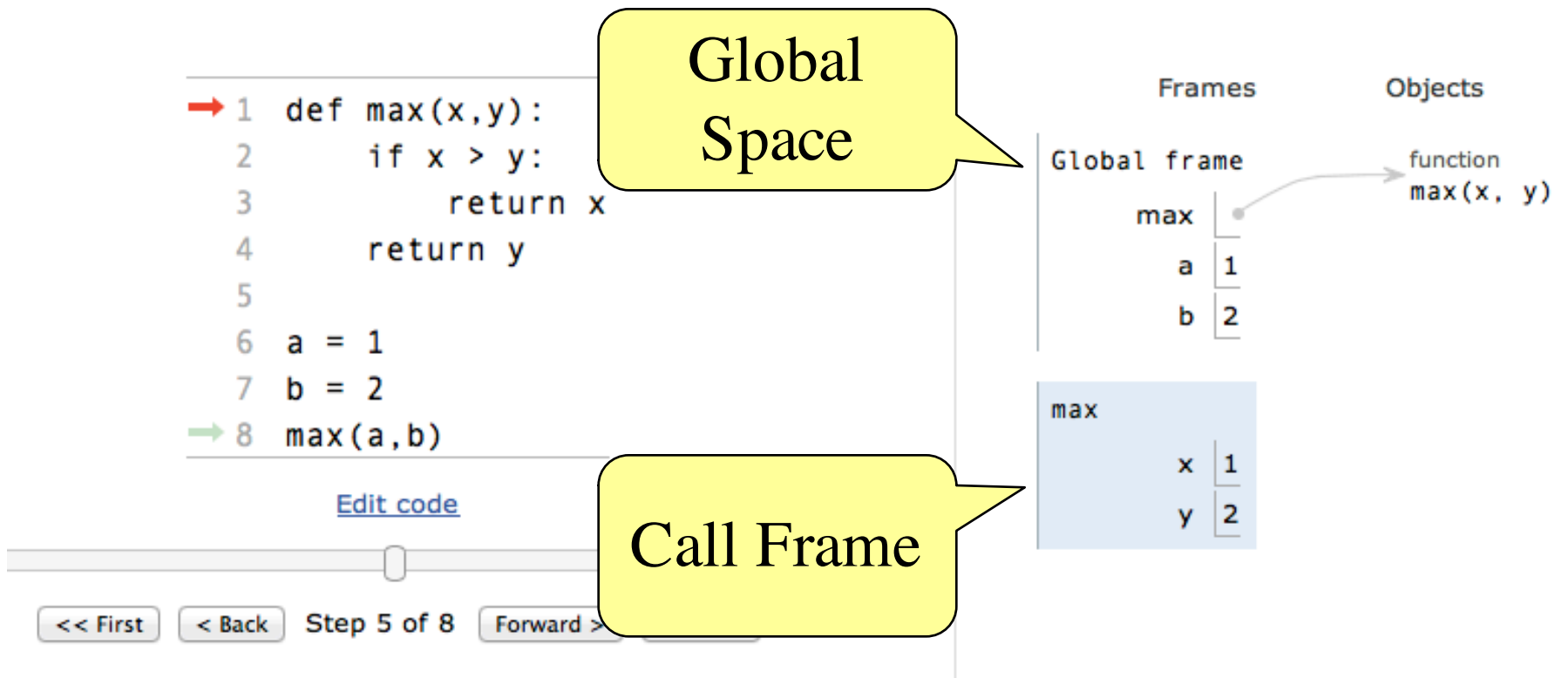
Step 5 of 8

Forward >

Last >>



Visualizing Frames: The Python Tutor



Visualizing Frames: The Python Tutor

The image shows a Python Tutor interface with the following code:

```
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)
```

The execution is at line 8. The frame visualization shows:

- Global Frame:** Contains variables `max` (pointing to the function `max(x, y)`), `a` (value 1), and `b` (value 2).
- Call Frame (max):** Contains local variables `x` (value 1) and `y` (value 2).

Annotations:

- A yellow callout bubble labeled "Global Space" points to the Global Frame.
- A yellow callout bubble labeled "Call Frame" points to the local frame for the `max` function.
- A green callout bubble labeled "Variables from second lecture go in here" points to the `max` variable in the Global Frame.

Navigation controls at the bottom include: << First, < Back, Step 5 of 8, Forward >, >> Last.

Visualizing Frames: The Python Tutor

```
→ 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)
```

[Edit code](#)

<< First

< Back

Step 5 of 8

Forward >

Last >>

Frames

Objects

Global fr

max

Missing line numbers!

a

b

max

x

1

y

2

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)
```

[Edit code](#)

<< First

< Back

Step 5 of 8

Forward >

Last >>

Frames

Objects

Global fr

max

Missing line
numbers!

a

b

max

x

1

y

2

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

Global Space
(for `globals.py`) a 4

show_a 1

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

def show_a():
    print a # shows 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

Global Space
(for `globals.py`) a 4

change_a

a 3.5

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

def change_a():
    a = 3.5 # local variable
```

Exercise Time

Function Definition

```
def foo(a,b):
```

```
    """Do something
```

```
       Param x: a number
```

```
       Param y: a number"""
```

```
1   x = a
```

```
2   y = b
```

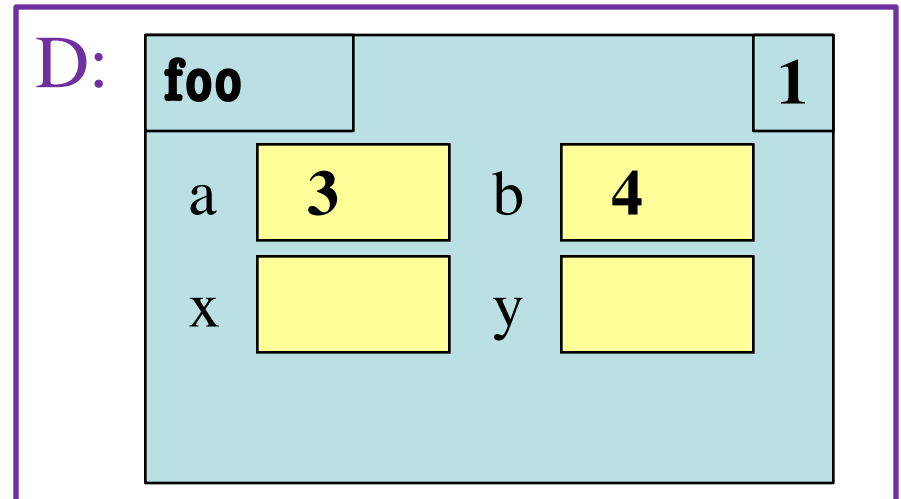
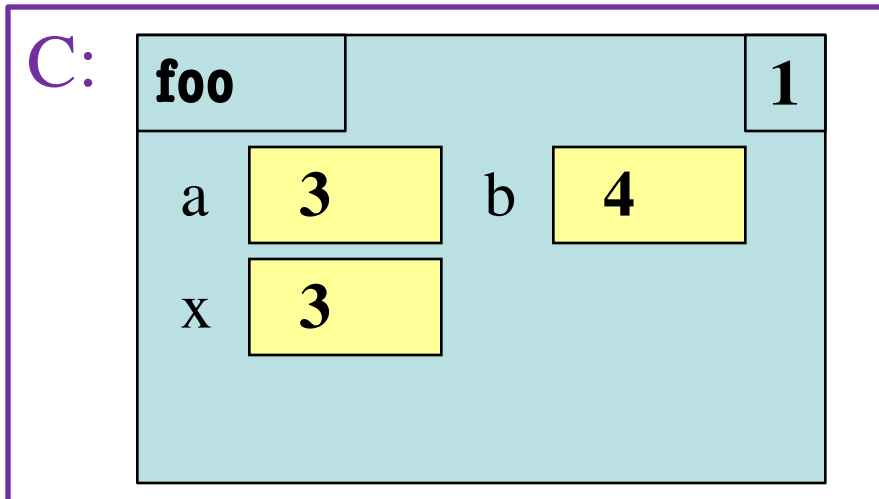
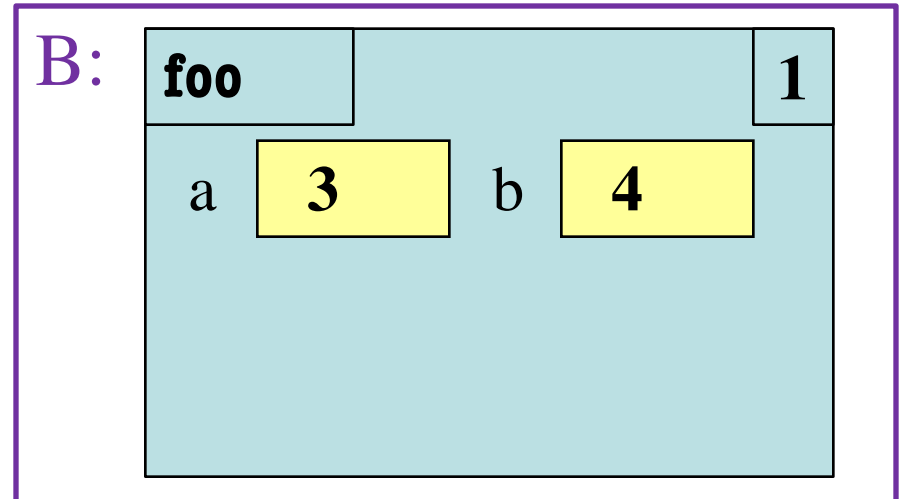
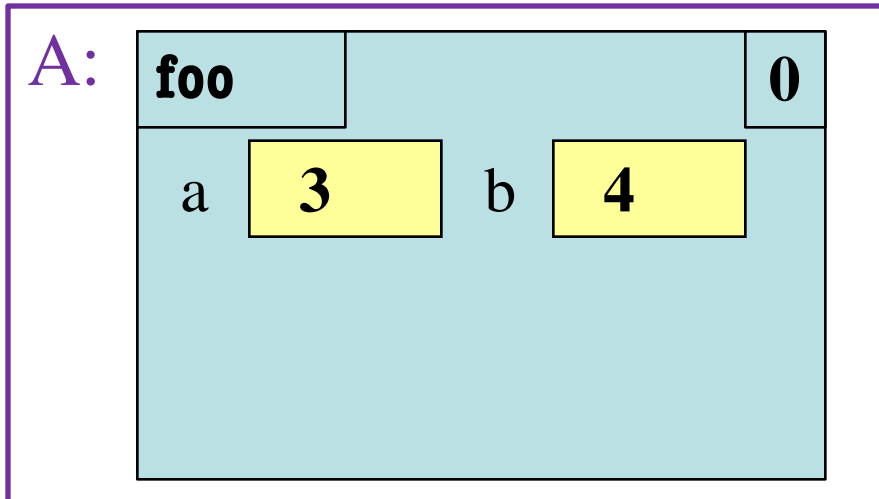
```
3   return x*y+y
```

Function Call

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

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

Which One is Closest to Your Answer?



Which One is Closest to Your Answer?

A:

foo		0	
a	3	b	4

B:

foo		1	
a	3	b	4

E:

—_ (ツ) _ /—

C:

foo	
a	3
x	3

		1	
b	4		
x		y	

Exercise Time

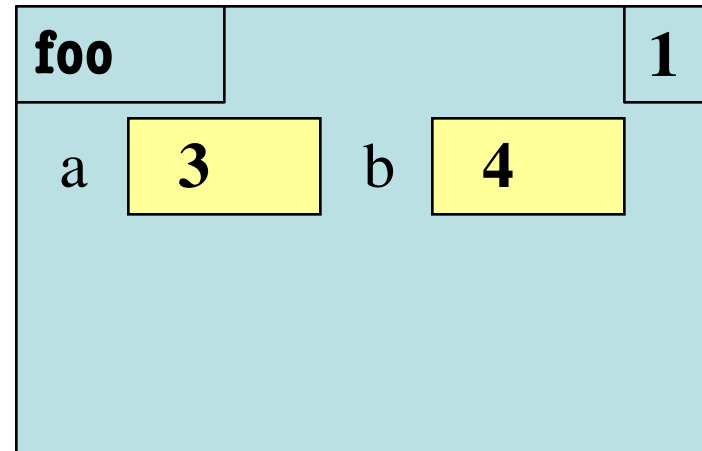
Function Definition

```
def foo(a,b):  
    """Do 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:



Exercise Time

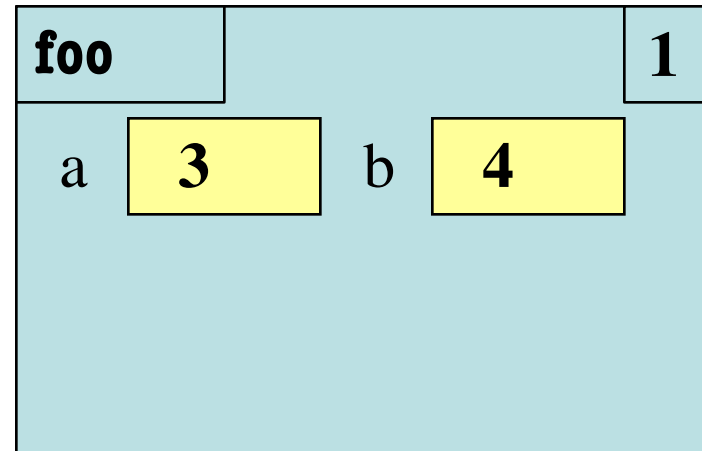
Function Definition

```
def foo(a,b):  
    """Do 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

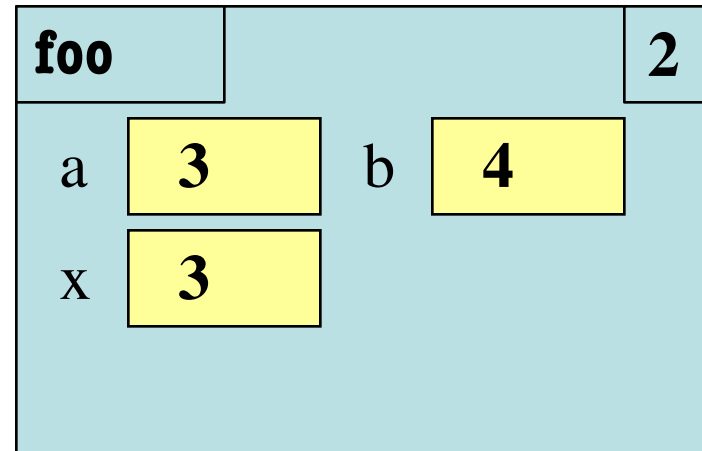
Function Definition

```
def foo(a,b):  
    """Do 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):
```

```
    """Do something
```

```
    Param x: a number
```

```
    Param y: a number"""
```

```
1 x = a
```

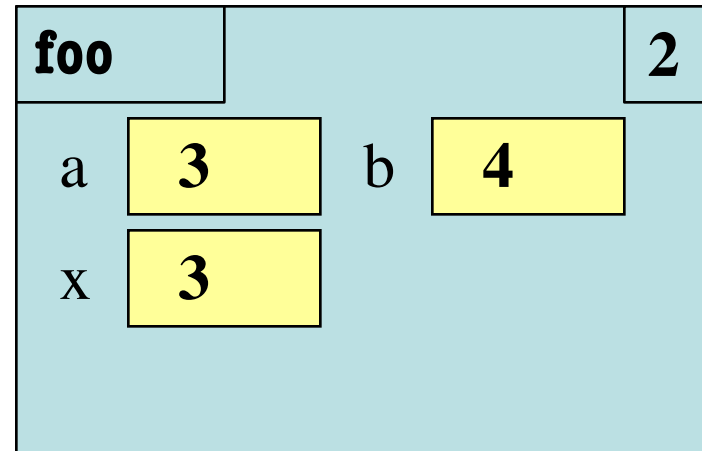
```
2 y = b
```

```
3 return x*y+y
```

Function Call

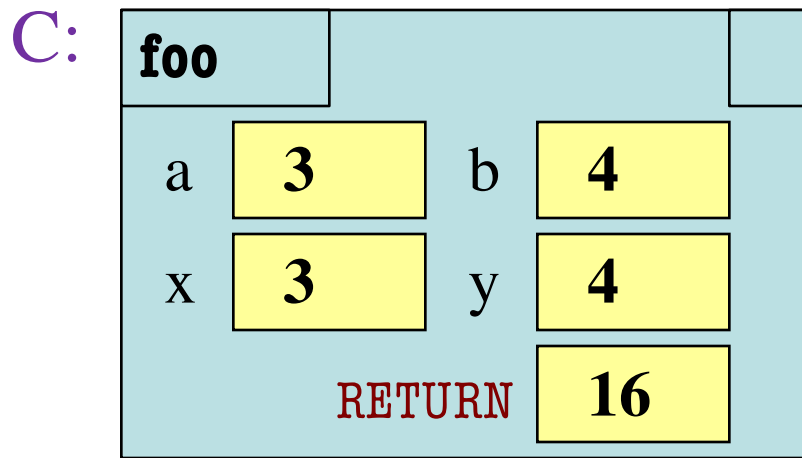
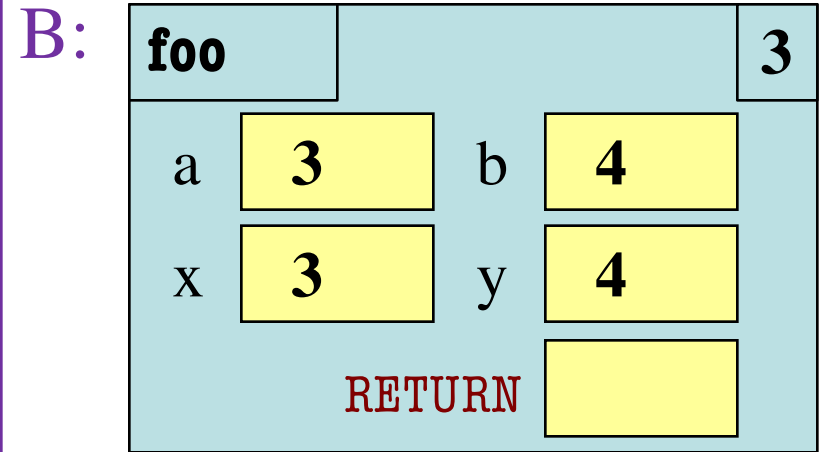
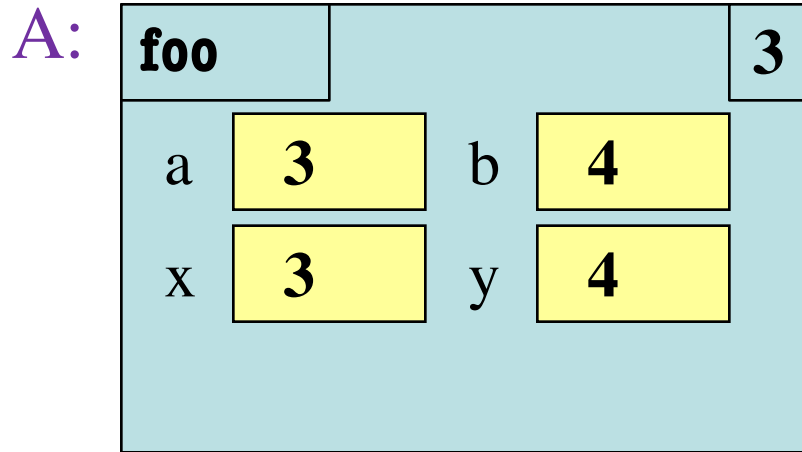
```
>>> x = foo(3,4)
```

C:



What is the **next step**?

Which One is Closest to Your Answer?



Exercise Time

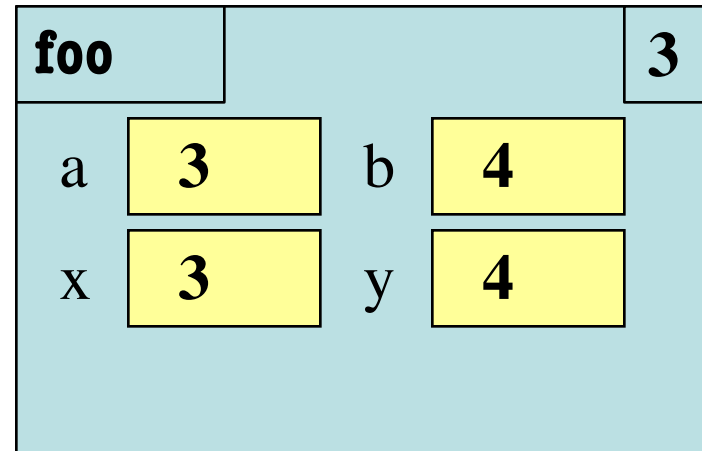
Function Definition

```
def foo(a,b):  
    """Do 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):
```

```
    """Do something
```

```
    Param x: a number
```

```
    Param y: a number"""
```

```
1 x = a
```

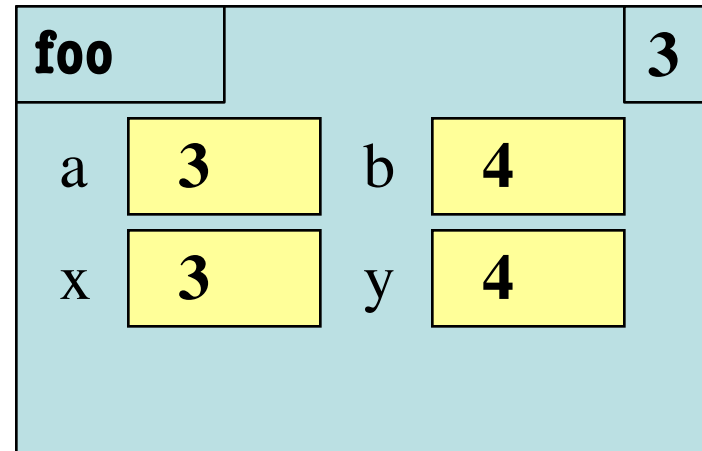
```
2 y = b
```

```
3 return x*y+y
```

Function Call

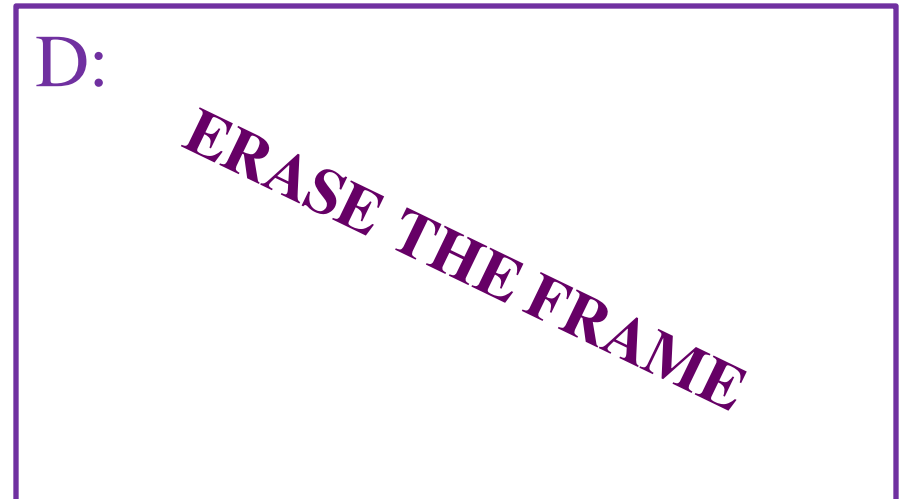
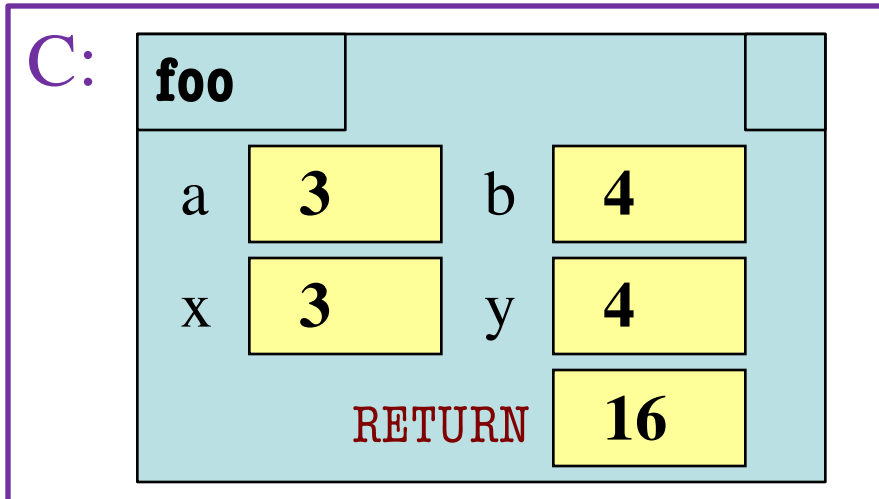
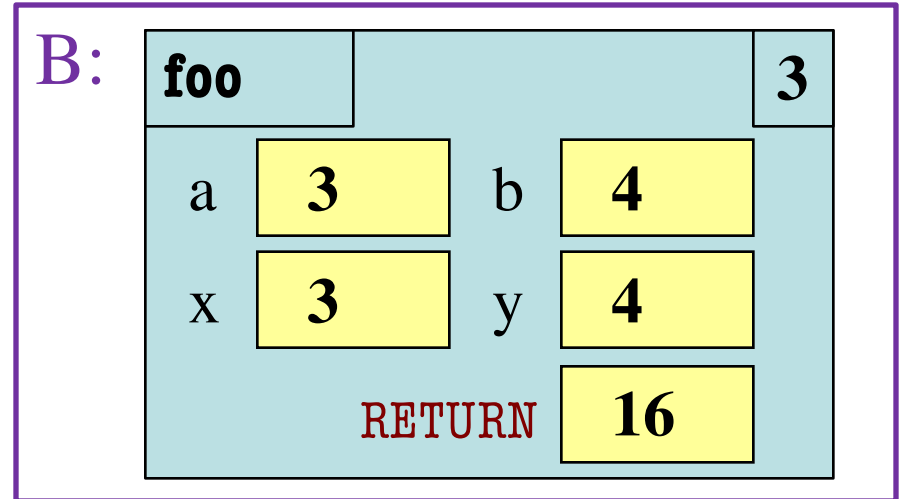
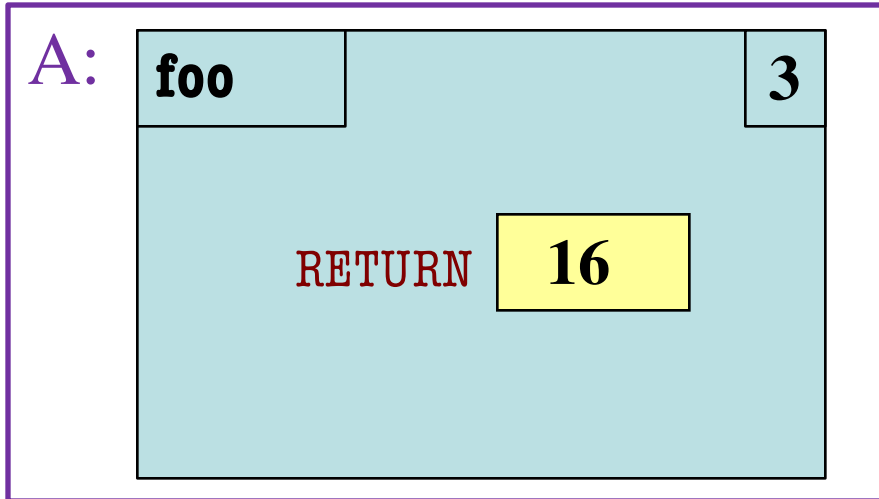
```
>>> x = foo(3,4)
```

A:



What is the **next step**?

Which One is Closest to Your Answer?



Exercise Time

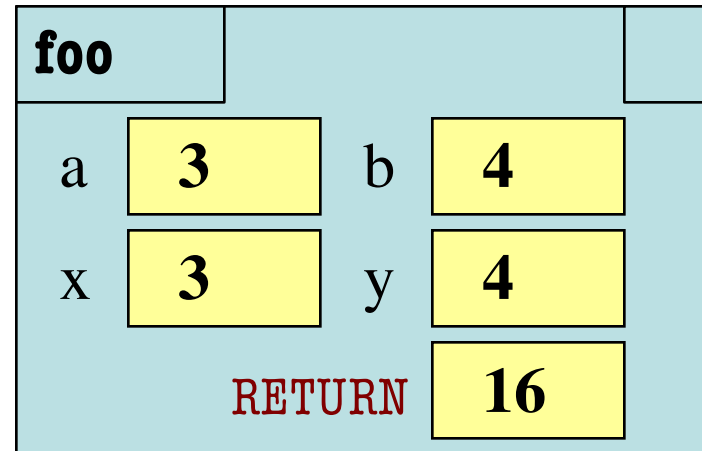
Function Definition

```
def foo(a,b):  
    """Do 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

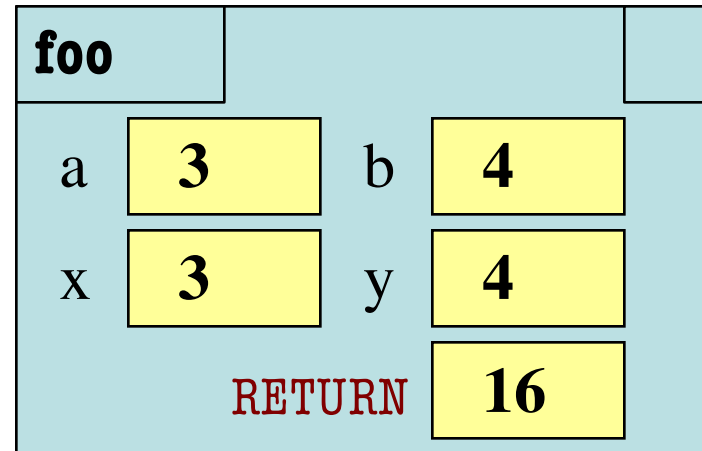
Function Definition

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

Function Call

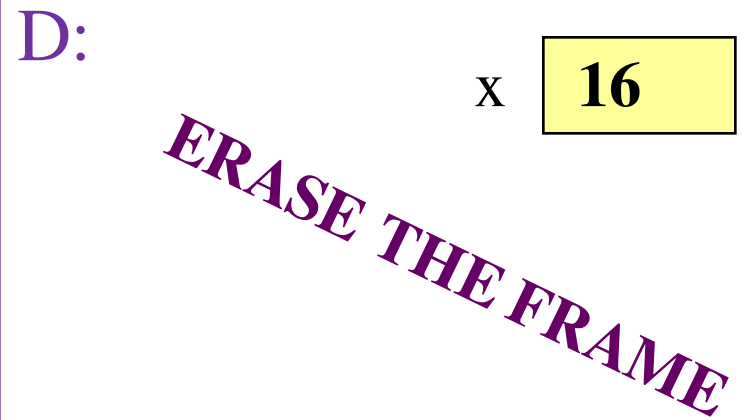
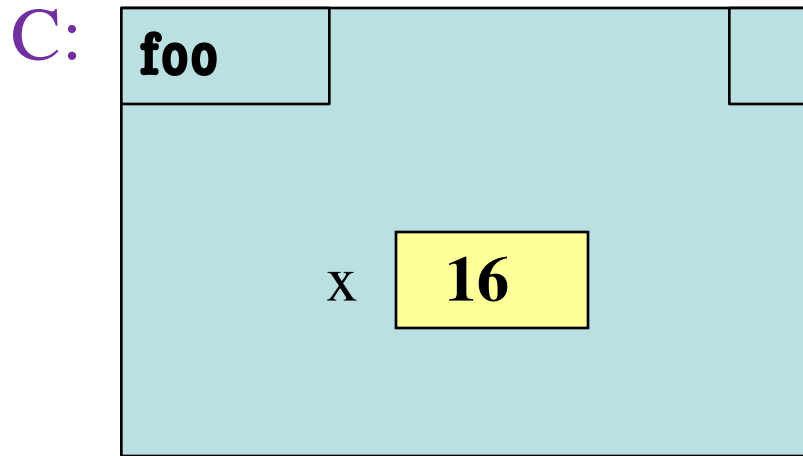
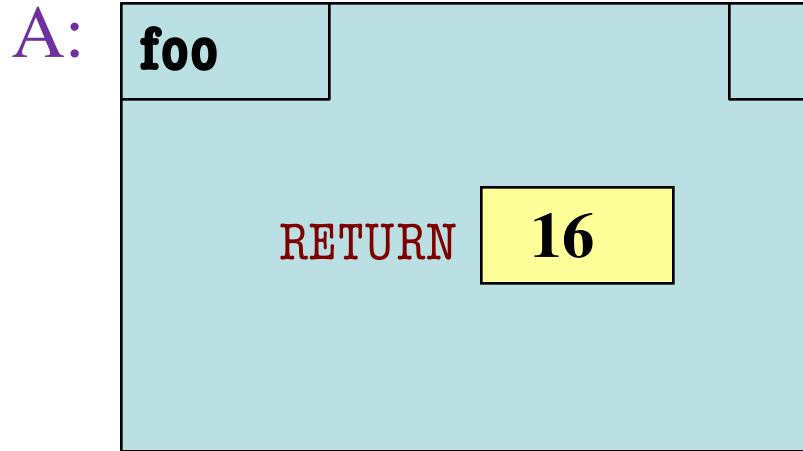
```
>>> x = foo(3,4)
```

C:



What is the **next step**?

Which One is Closest to Your Answer?



Exercise Time

Function Definition

```
def foo(a,b):  
    """Do 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)
```

D:

x

16

ERASE THE FRAME

Exercise Time

Function Definition

```
def foo(a,b):  
    """Do 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)
```

D:

Variable in
global space

x

16

ERASE THE FRAME