CS 1110:

Introduction to Computing Using Python

Lecture 4

Defining Functions

[Andersen, Gries, Lee, Marschner, Van Loan, White]

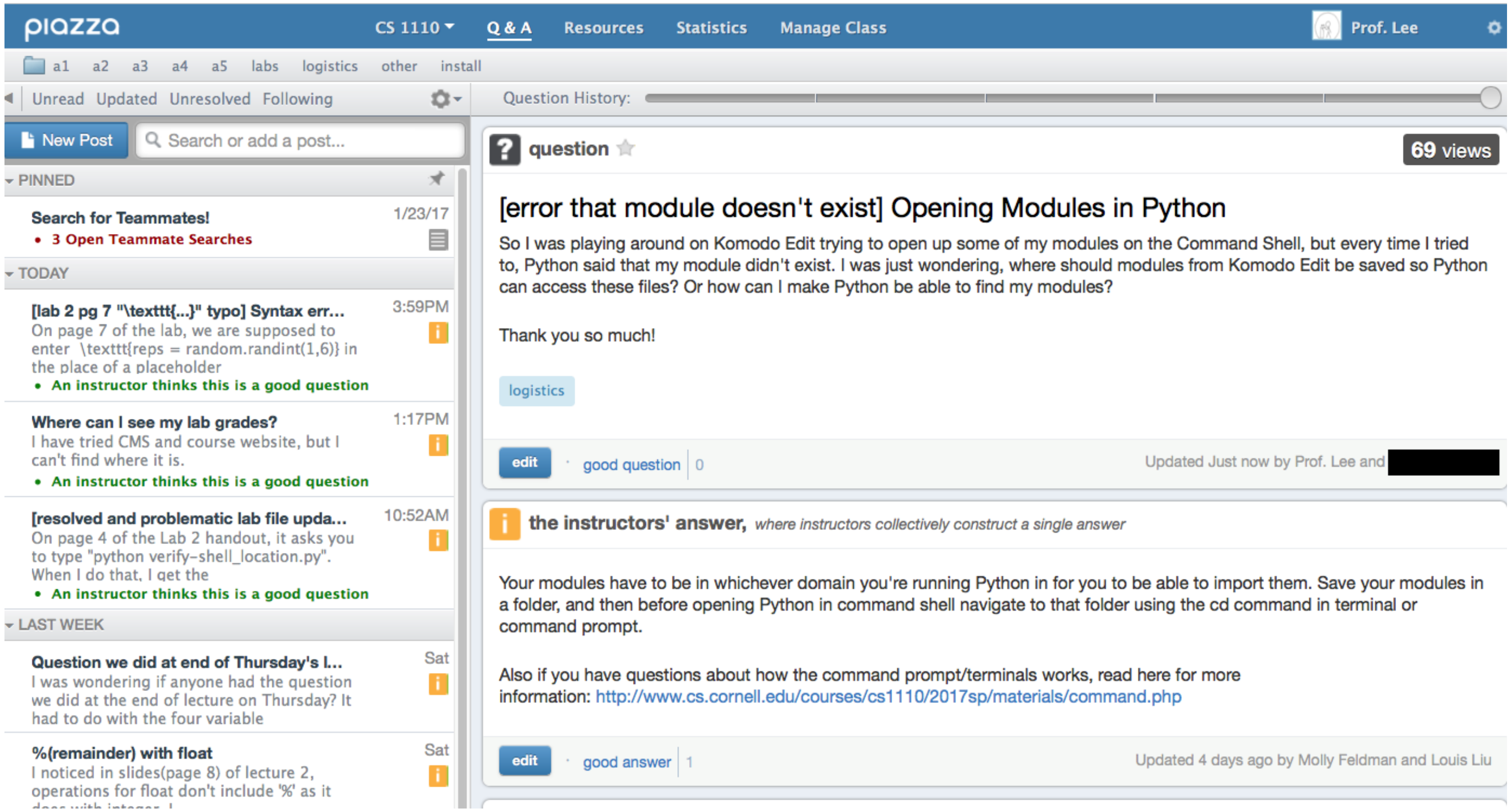
Things to Do Before Next Class

- Read the textbook:
  - Chapter 8.1, 8.2, 8.4, 8.5, first paragraph of 8.9
- Go to lab

Lab Website

- Can see if you’ve gotten credit for labs
- https://cs1110.cs.cornell.edu/labs/

Piazza



Search for Teammates! 1/23/17
• 3 Open Teammate Searches

[lab 2 pg 7 "\texttt{...}" typo] Syntax err... 3:59PM
On page 7 of the lab, we are supposed to enter \texttt{reps = random.randint(1,6)} in the place of a placeholder
• An instructor thinks this is a good question

Where can I see my lab grades? 1:17PM
I have tried CMS and course website, but I can't find where it is.
• An instructor thinks this is a good question

[resolved and problematic lab file upda... 10:52AM
On page 4 of the Lab 2 handout, it asks you to type "python verify-shell_location.py". When I do that, I get the
• An instructor thinks this is a good question

Question we did at end of Thursday's l... Sat
I was wondering if anyone had the question we did at the end of lecture on Thursday? It had to do with the four variable

%(remainder) with float Sat
I noticed in slides(page 8) of lecture 2, operations for float don't include '%' as it

question — 69 views

[error that module doesn't exist] Opening Modules in Python

So I was playing around on Komodo Edit trying to open up some of my modules on the Command Shell, but every time I tried to, Python said that my module didn't exist. I was just wondering, where should modules from Komodo Edit be saved so Python can access these files? Or how can I make Python be able to find my modules?

Thank you so much!

logistics

edit · good question | 0 — Updated Just now by Prof. Lee and [redacted]

the instructors' answer, *where instructors collectively construct a single answer*

Your modules have to be in whichever domain you're running Python in for you to be able to import them. Save your modules in a folder, and then before opening Python in command shell navigate to that folder using the cd command in terminal or command prompt.

Also if you have questions about how the command prompt/terminals works, read here for more information: http://www.cs.cornell.edu/courses/cs1110/2017sp/materials/command.php

edit · good answer | 1 — Updated 4 days ago by Molly Feldman and Louis Liu

From last time: Function Calls

- Function expressions have the form $\underbrace{\textbf{fun}}_{\text{function name}}(\underbrace{x}_{\text{argument}},y,\ldots)$

- **Examples** (math functions that work in Python):
  - round(2.34)
  - max(a+3,24)

From last time: Modules

- Modules provide extra functions, variables
  - Access them with the import command
- **Example**: module math

```
>>> import math
>>> math.cos(2.0)
-0.4161468365471424
>>> math.pi
3.141592653589793
```

From last time: Modules

Module Text

```
# module.py

"""This is a simple module.
It shows how modules work"""

x = 1+2
x = 3*x
```

Python Command Shell

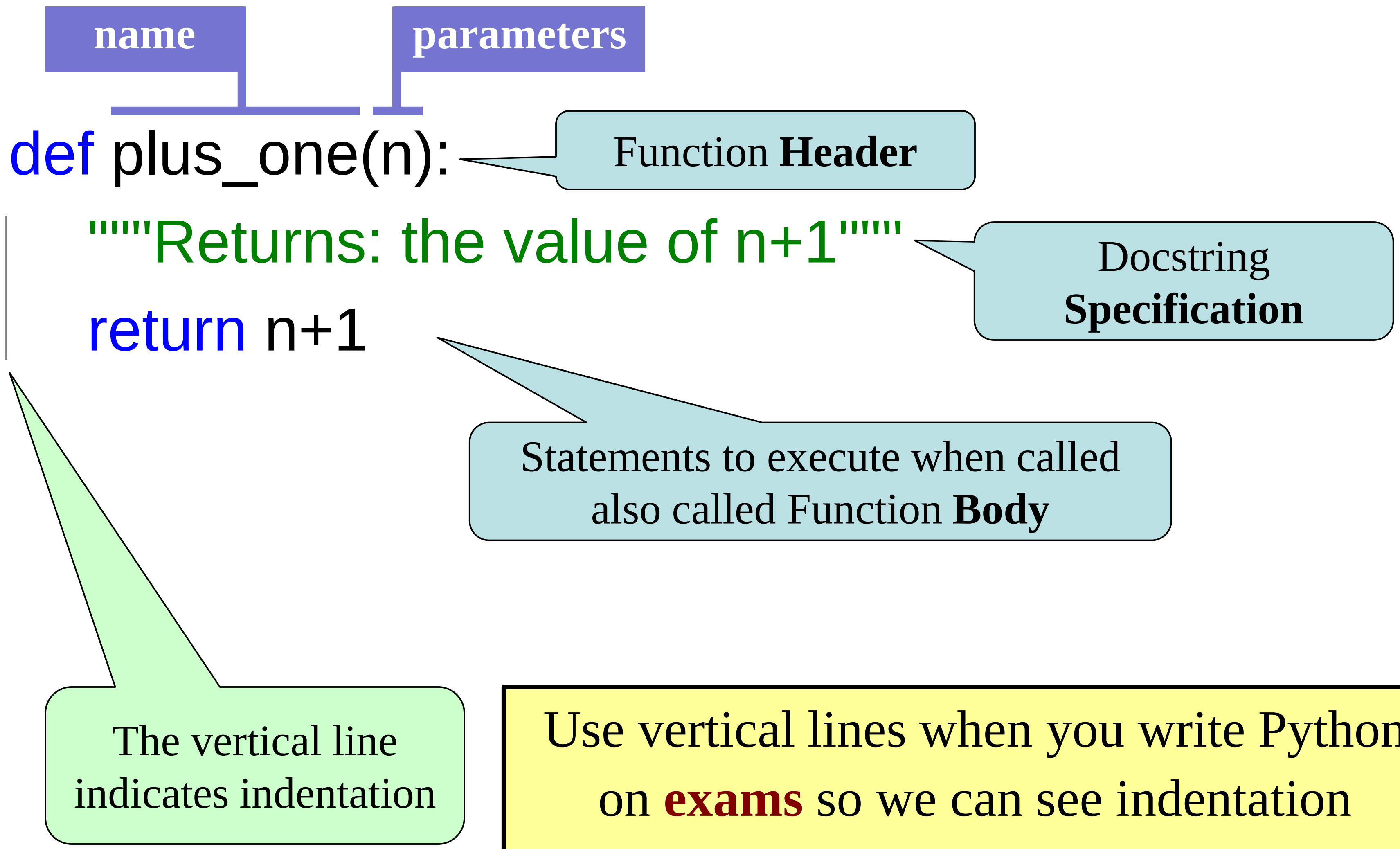
```
>>> import module
>>> module.x
9
```

- We discussed how to make module *variables*
- Have not covered how to make *functions*

increment.py

```
>>> import increment
>>> increment.plus_one(1)
2
>>> increment.plus_one(2)
3
```

Anatomy of a Function Definition



```
def plus_one(n):
    """Returns: the value of n+1"""
    return n+1
```

name
parameters
Function **Header**
Docstring **Specification**
Statements to execute when called also called Function **Body**
The vertical line indicates indentation

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

Function Calls vs. Definitions

Function Call

- Command to **do** the function

```
>>> increment.plus_one(23)
24
>>>
```

argument to assign to n

Function Definition

- Defines what function **does**

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

declaration of **parameter** n

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

Using increment.py

Module Text

```
# increment.py

"""Increment function module"""

def plus_one(n):
    """Returns: n+1"""
    return n+1
```

Python Command Shell

```
>>> import increment
```

Python skips

Python skips

Python learns the function definition

Python skips everything inside the function

Using increment.py

Module Text

```
# increment.py

"""Increment function module"""

def plus_one(n):
    """Returns: n+1"""
    return n+1
```

Now Python executes the function body

Python Command Shell

```
>>> import increment
>>> increment.plus_one(23)
```

Python knows what this is!

Using increment.py

Module Text

```python
# increment.py

"""Increment function module"""

def plus_one(n):
    """Returns: n+1"""
    return n+1
```

Python Command Shell

```
>>> import increment
>>> increment.plus_one(23)
```

Using increment.py

Module Text

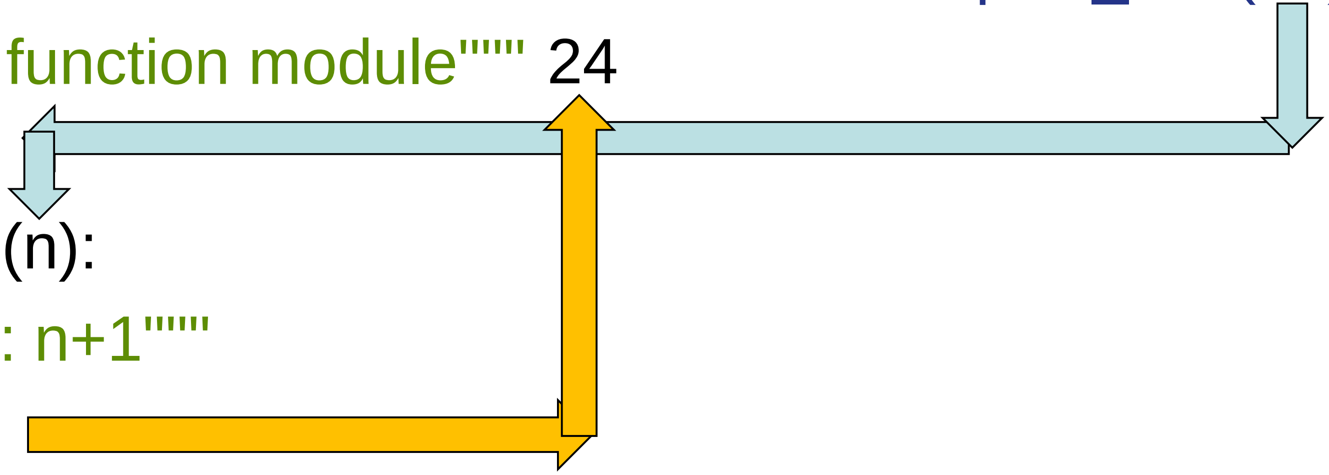
```python
# increment.py

"""Increment function module"""

def plus_one(n):
    """Returns: n+1"""
    return n+1
```

Python Command Shell

```
>>> import increment
>>> increment.plus_one(23)
24
```



The return Statement

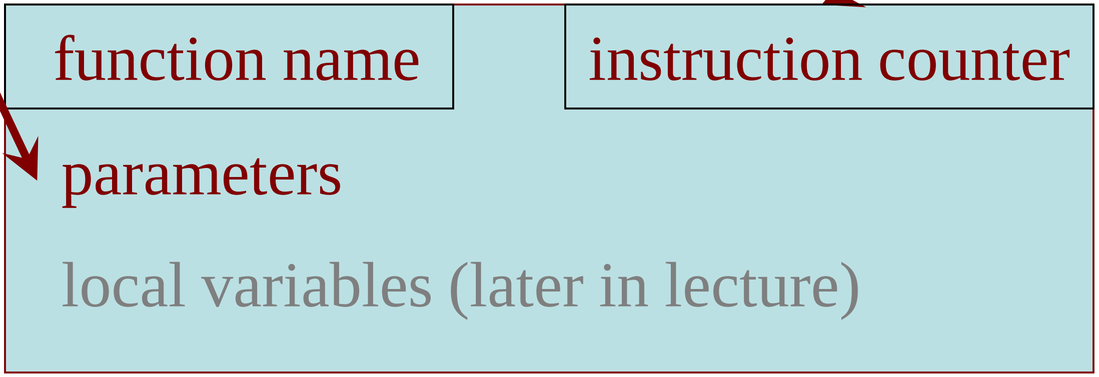
- Passes a value from the function to the caller
- **Format**: **return** *<expression>*
- Any statements after **return** are ignored
- Optional (if absent, no value will be sent back)

Understanding How Functions Work

- We will draw pictures to show what is in memory
- **Function Frame**: Representation of function call

Draw parameters as variables (named boxes)

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



Example: to_centigrade

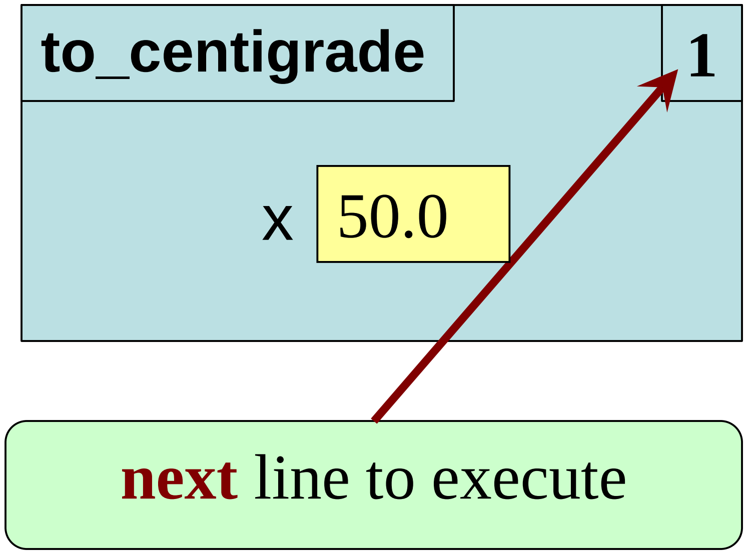
```
>>> from temperature import *
>>> to_centigrade(50.0)
```

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

Example: to_centigrade(50.0)

PHASE 1: Set up call frame

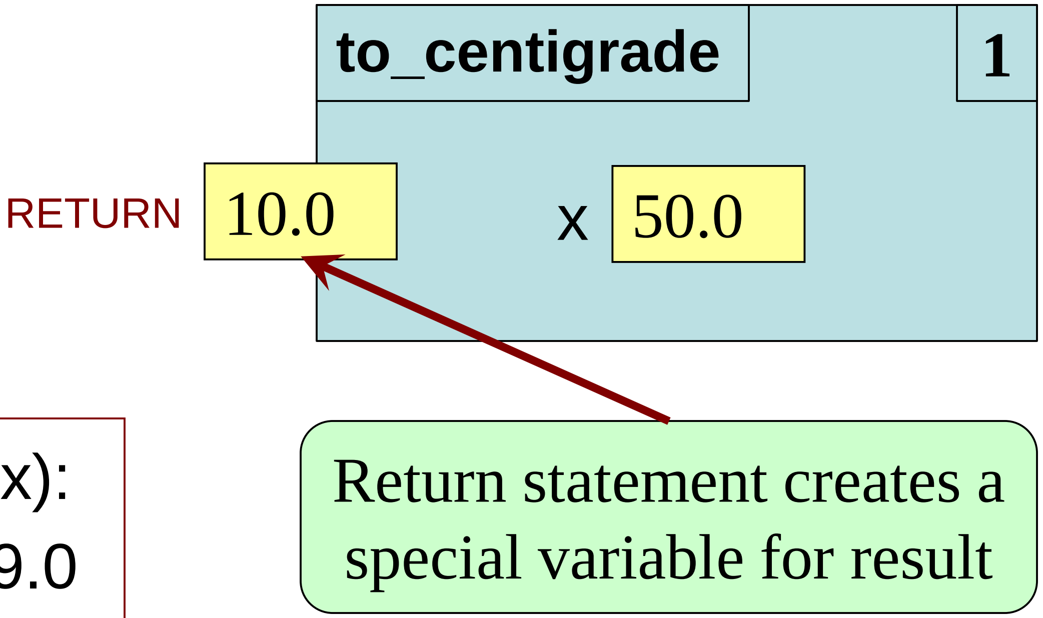
1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Indicate next line to execute



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

Example: to_centigrade(50.0)

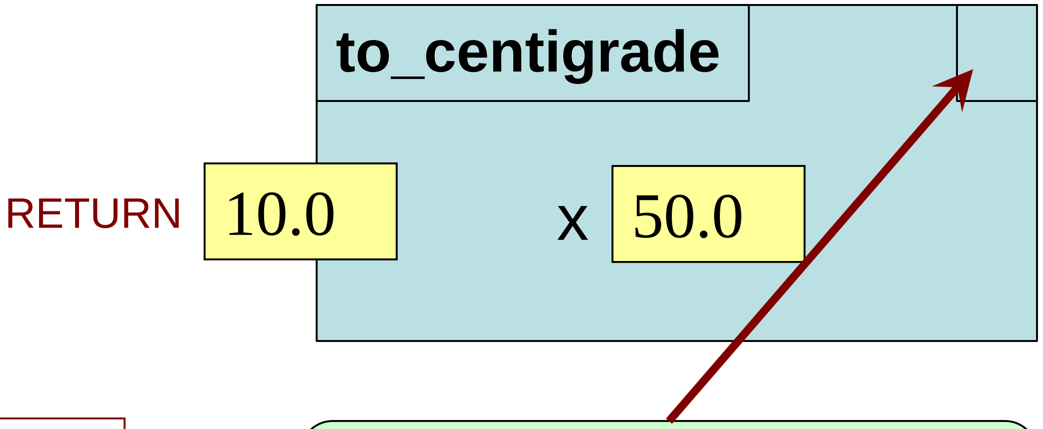
PHASE 2: Execute function body



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

Example: to_centigrade(50.0)

PHASE 2: Execute function body

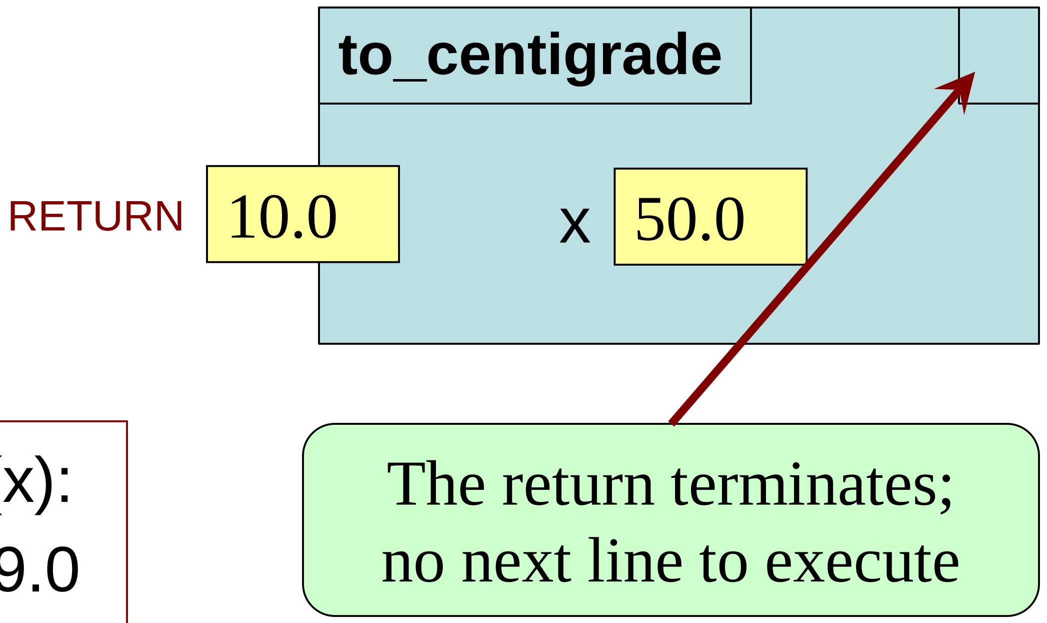


The return terminates; no next line to execute

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

Example: to_centigrade(50.0)

PHASE 3: Erase call frame



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

Example: to_centigrade(50.0)

PHASE 3: Erase call frame

ERASE WHOLE FRAME

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

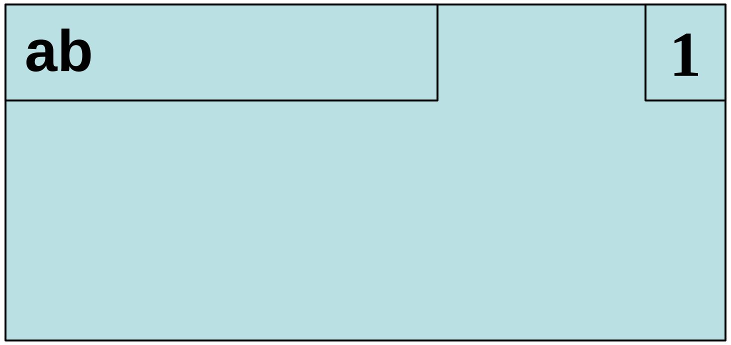
But don’t actually erase on an exam

Local Variables

- Call frames can make “local” variables

```
>>> import variables
>>> variables.ab()
```

```
   def ab():
1      a = 1
2      b = 2
```

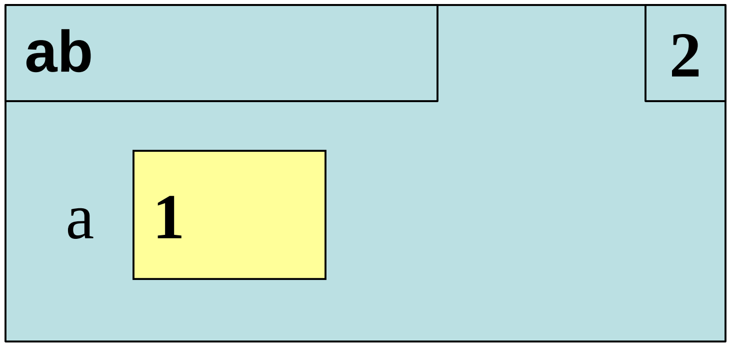


Local Variables

- Call frames can make "local" variables

```
>>> import variables
>>> variables.ab()
```

```
  def ab():
1     a = 1
2     b = 2
```

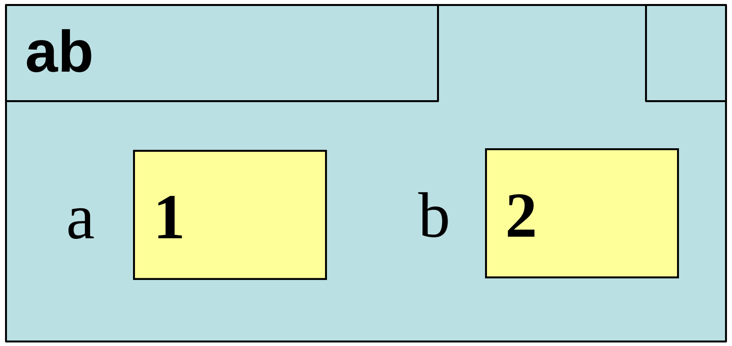


Local Variables

- Call frames can make "local" variables

```
>>> import variables
>>> variables.ab()
```

```
  def ab():
1     a = 1
2     b = 2
```



Local Variables

- Call frames can make "local" variables

```
>>> import variables
>>> variables.ab()
```

```
def ab():
1   a = 1
2   b = 2
```

ERASE WHOLE FRAME

Variables are gone! This function is useless.

Exercise Time

Function Definition

```
def foo(a,b):
    x = a
    y = b
    return x*y+y
```

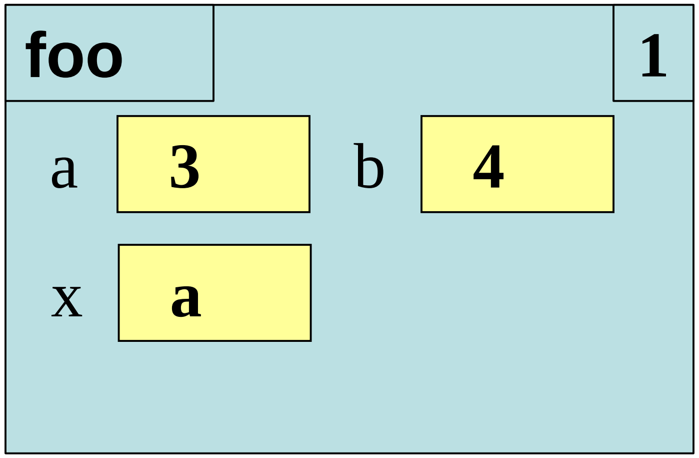
Function Call

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

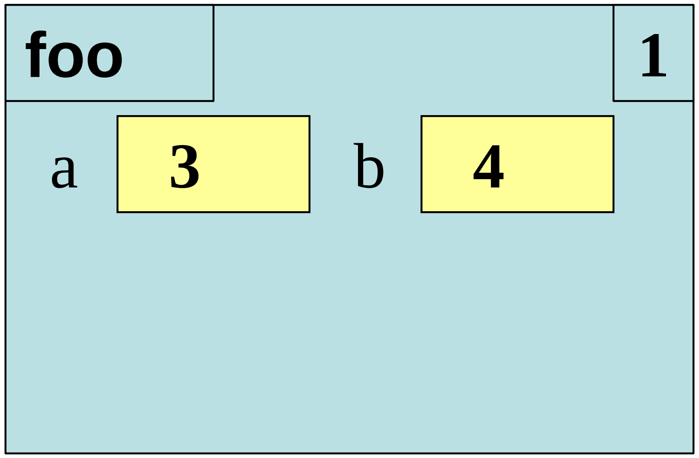
What does the frame look like at the **start**?

Which One is Closest to Your Answer?

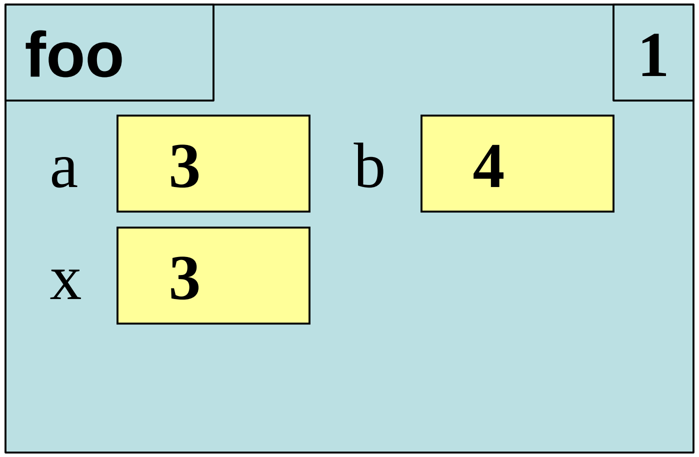
A:



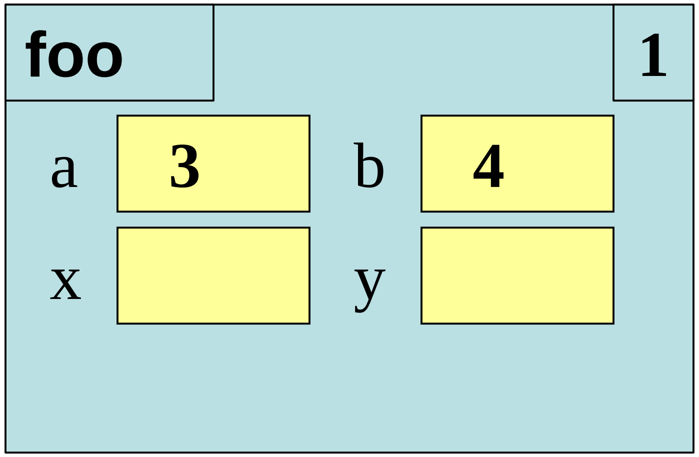
B:



C:



D:



Exercise Time

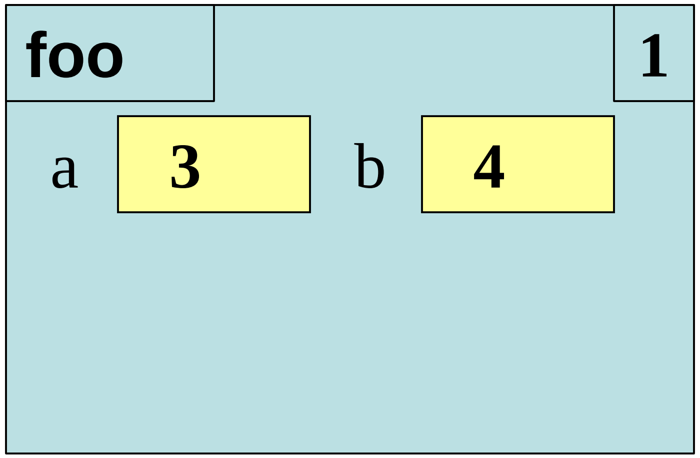
Function Definition

```
def foo(a,b):
1     x = a
2     y = b
3     return x*y+y
```

Function Call

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

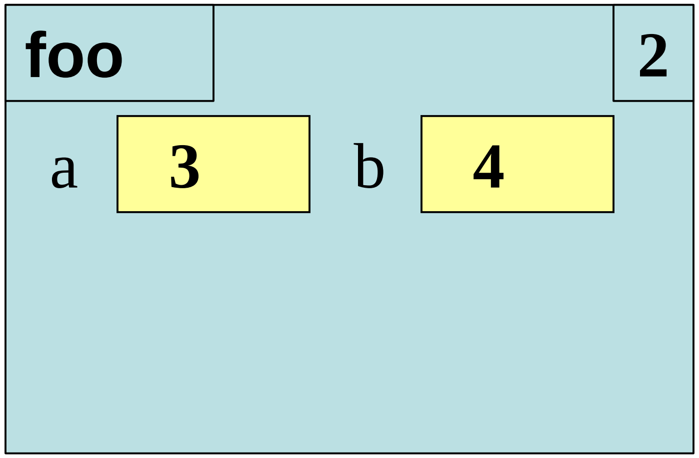
B:



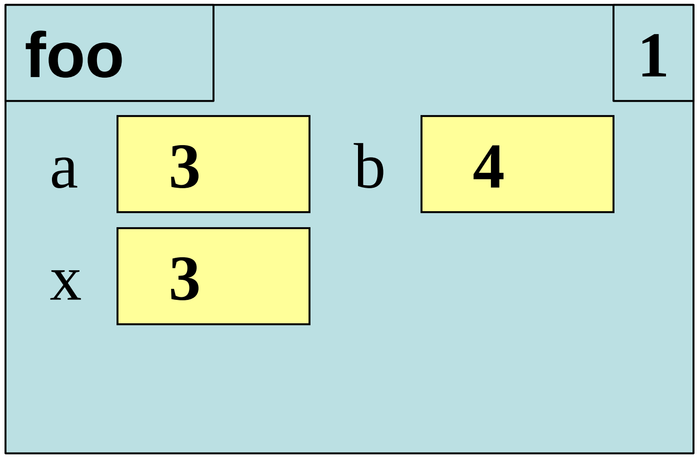
What is the **next step**?

Which One is Closest to Your Answer?

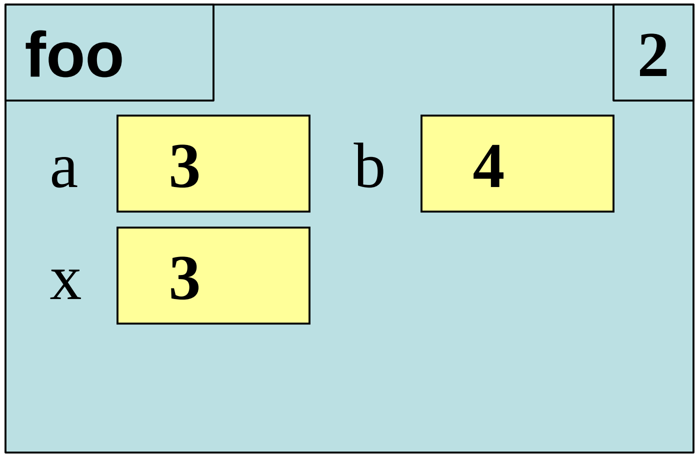
A:



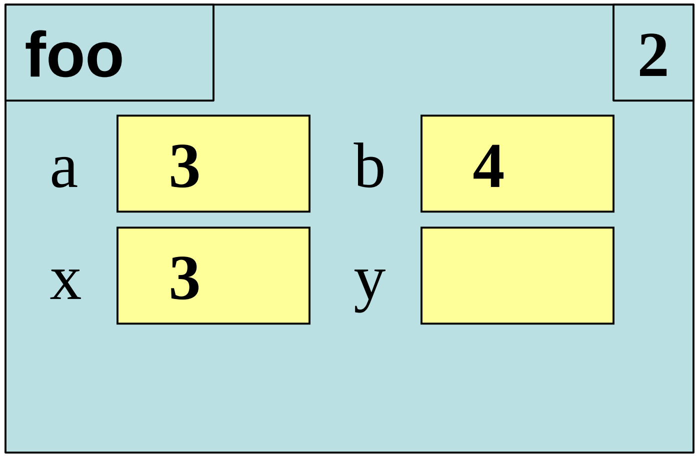
B:



C:



D:



Exercise Time

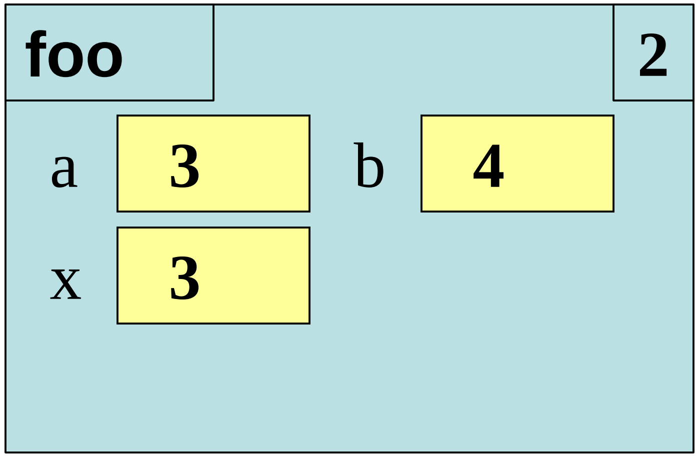
Function Definition

```
def foo(a,b):
1    x = a
2    y = b
3    return x*y+y
```

Function Call

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

C:



What is the **next step**?

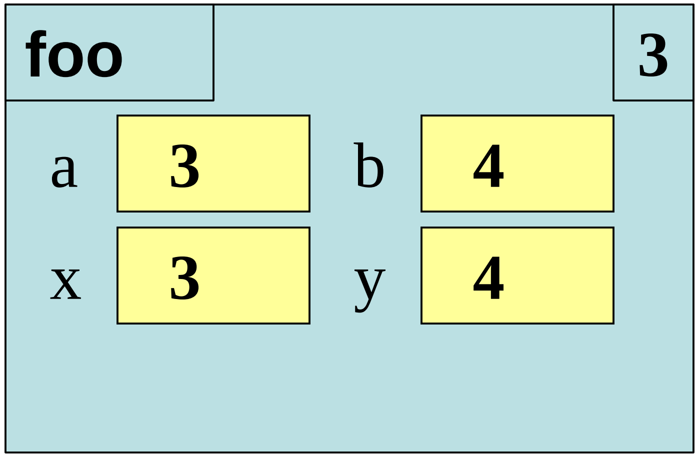
Exercise Time

Function Definition

```
def foo(a,b):
1       x = a
2       y = b
3       return x*y+y
```

Function Call

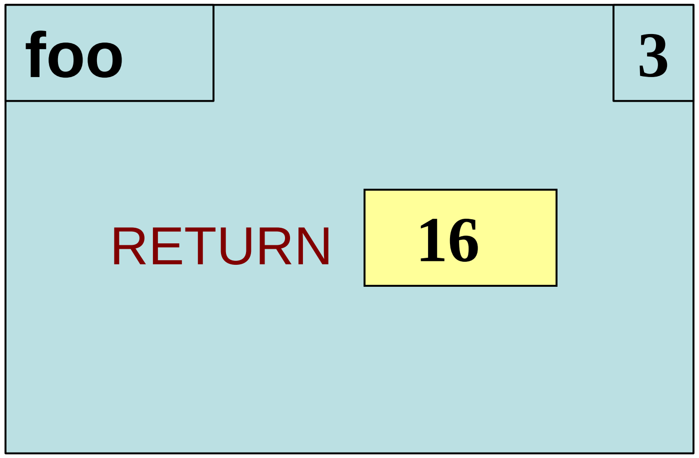
```
>>> foo(3,4)
```



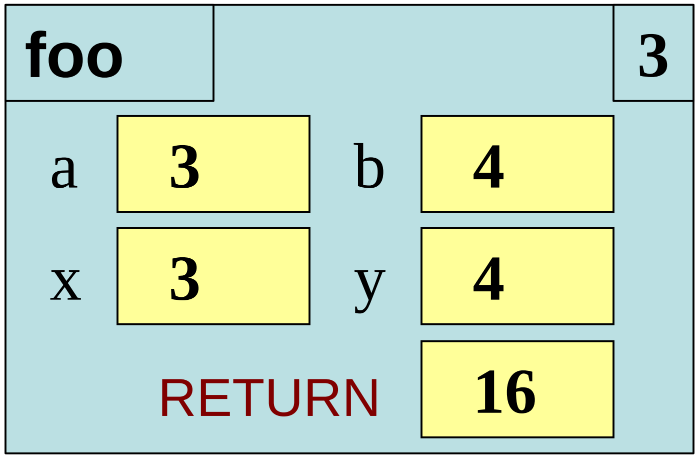
What is the **next step**?

Which One is Closest to Your Answer?

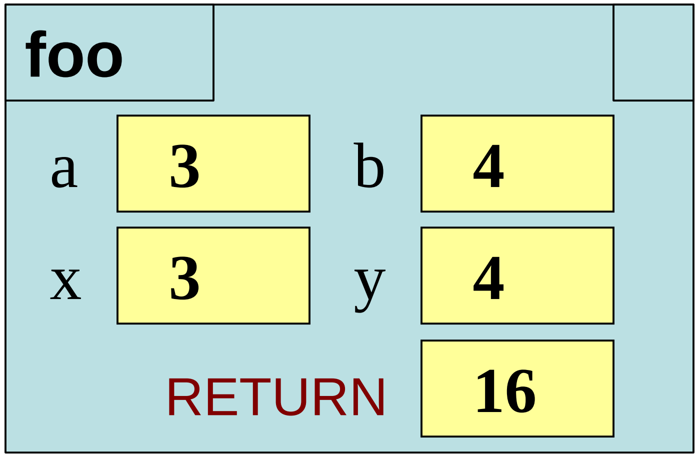
A:



B:



C:



D:

ERASE THE FRAME

Exercise Time

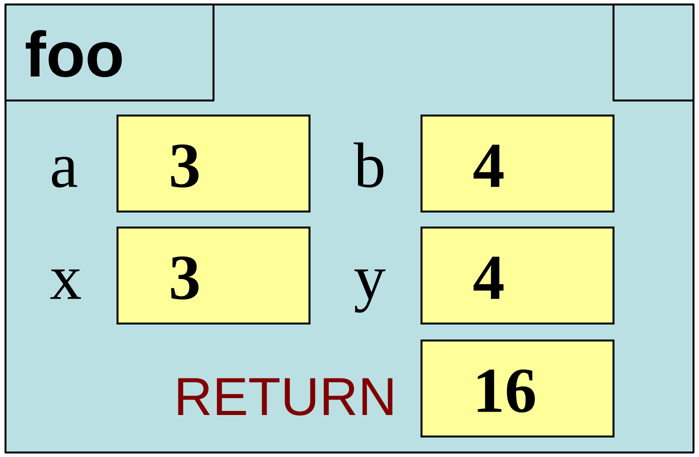
Function Definition

```
def foo(a,b):
1     x = a
2     y = b
3     return x*y+y
```

Function Call

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

C:



What is the **next step**?

Exercise Time

Function Definition

```
def foo(a,b):
    x = a
    y = b
    return x*y+y
```

Function Call

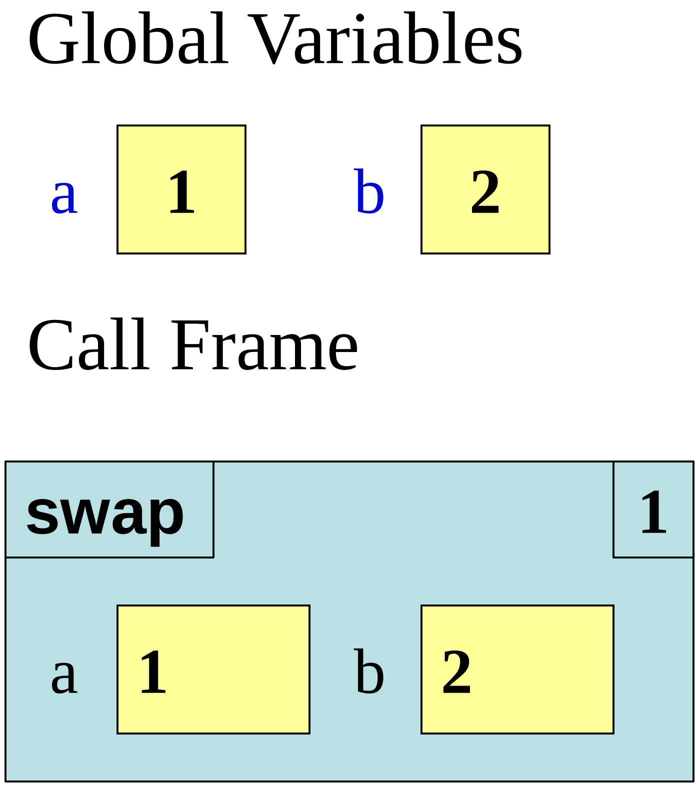
```
>>> foo(3,4)
>>> 16
```

ERASE THE FRAME

Call Frames and Global Variables

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

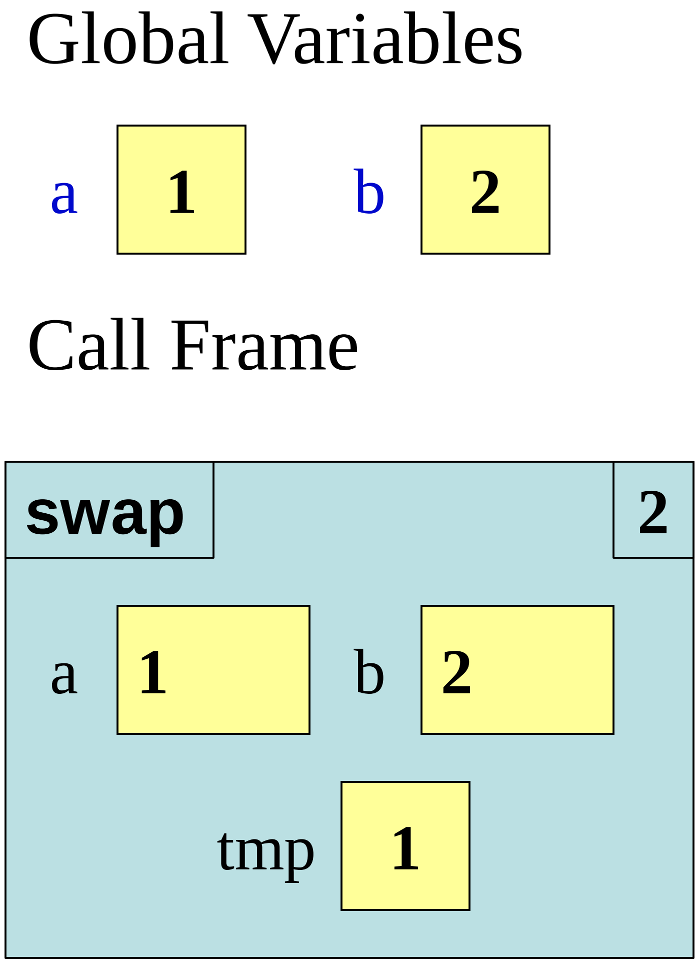
```
>>> a = 1
>>> b = 2
>>> swap(a,b)
```



Call Frames and Global Variables

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

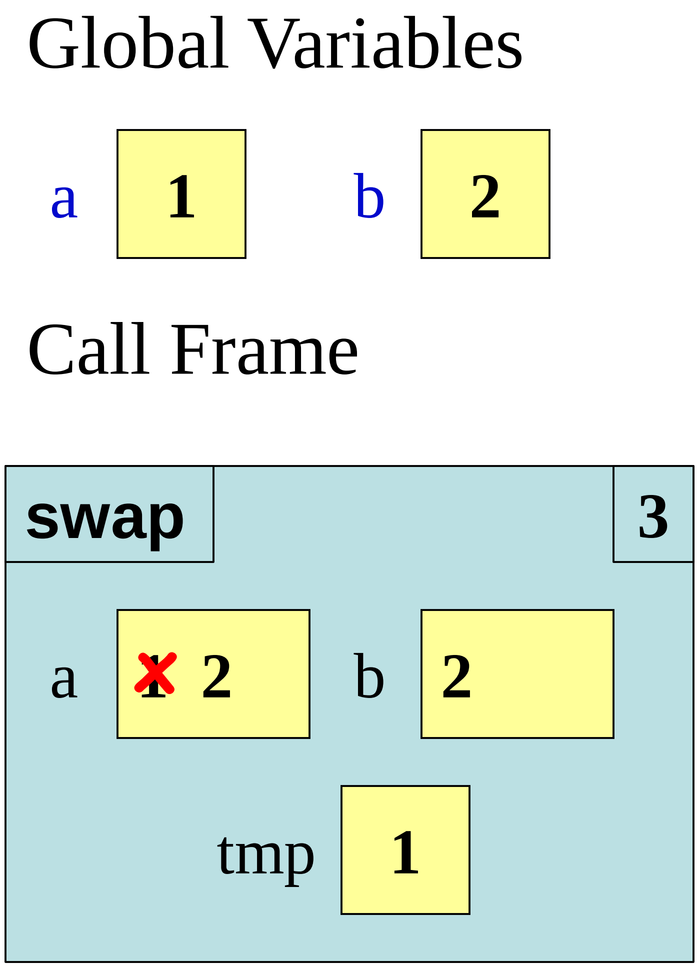
```
>>> a = 1
>>> b = 2
>>> swap(a,b)
```



Call Frames and Global Variables

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

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

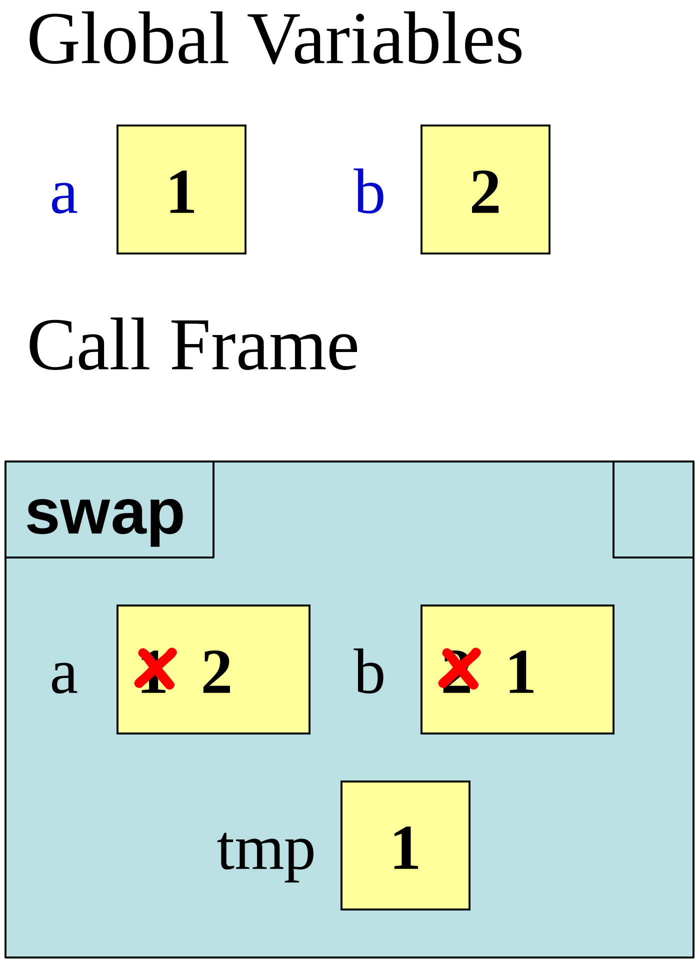


Call Frames and Global Variables

```
    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

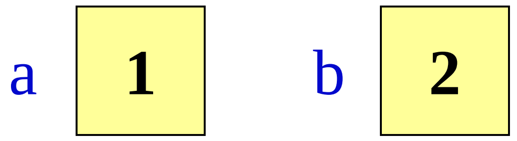


Call Frames and Global Variables

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

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

Global Variables



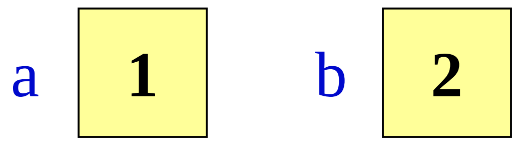
Call Frame

ERASE THE FRAME

Call Frames and Global Variables

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

Global Variables



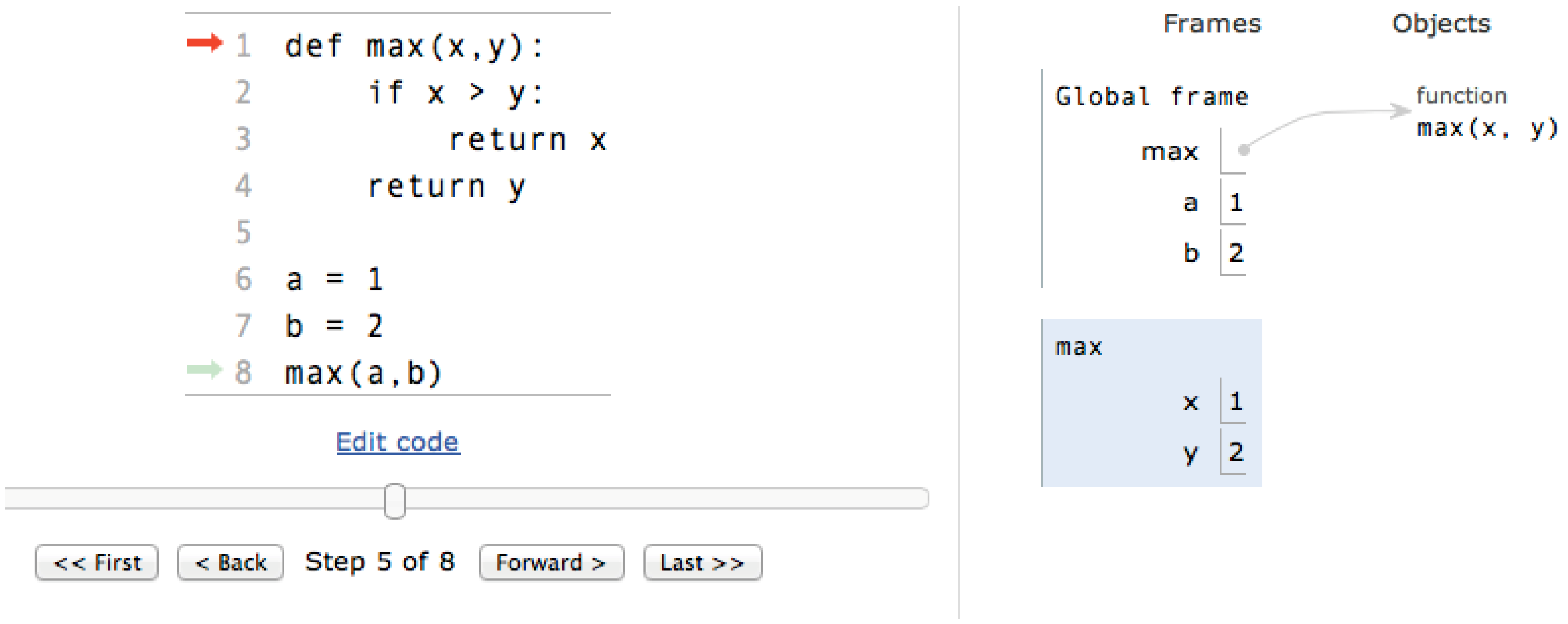
Call F

THIS FUNCTION DOES NOT SWAP
the *global* a and *global* b

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

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)
```

More Exercises

Module Text	Python Command Shell

```
# module.py

def foo(x):
    return x+1

x = 1+2
x = 3*x
```

```
>>> import module
>>> module.x
…
```

What does Python give me?

A: 9 **CORRECT**
B: 10
C: 1
D: Nothing
E: Error