http://www.cs.cornell.edu/courses/cs1110/2019sp

Lecture 4: Defining Functions

(Ch. 3.4-3.11)

CS 1110

Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

From last time: Function Calls

- Function expressions have the form **fun**(x,y,…)

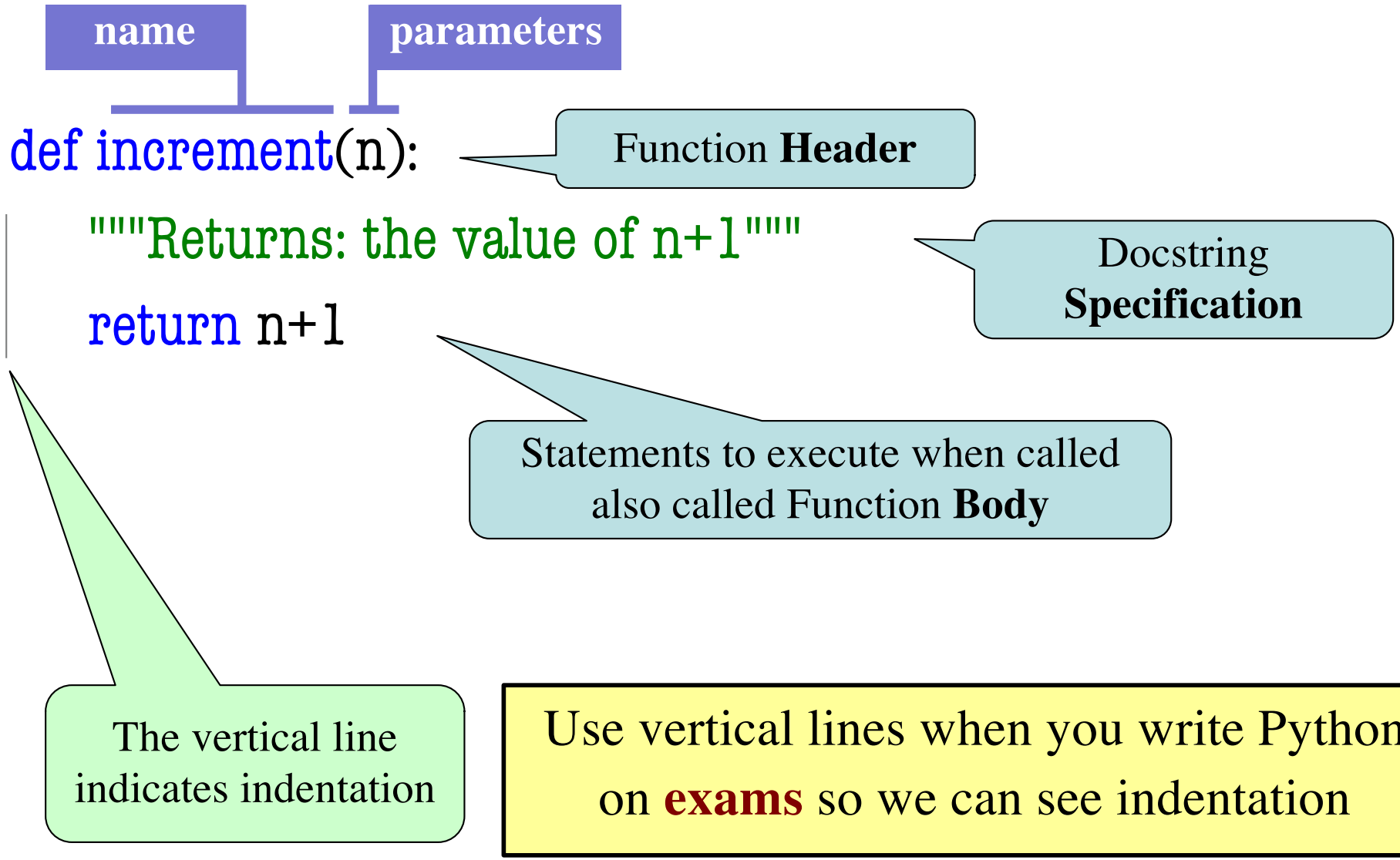
function name

argument

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

Let's define our own functions!

Anatomy of a Function Definition



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

The **`return`** Statement

- Passes a value from the function to the caller
- **Format**: **`return`** *<expression>*
- Any statements after **`return`** are ignored
- Optional (if absent, special value **`None`** will be sent back)

Function Definitions vs. Calls

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


increment(2)
```

simple_math.py

Function definition

- Defines what function **does**
- Declaration of parameter **n**
- **Parameter:** the variable that is listed within the parentheses of a function header.

Function call

- Command to do the function
- Argument to assign to **n**
- Argument: a value to assign to the function parameter when it is called

Executing the script simple_math.py

```
# simple_math.py

"""script that defines and
calls one simple math
function"""”

def increment(n):
    """Returns: n+1"""
    return n+1

increment(2)
```

Python skips

Python skips

Python learns about the function

Python skips everything inside the function

Python executes this statement

Show in python tutor
(put an error inside fn)

Now Python executes the function body

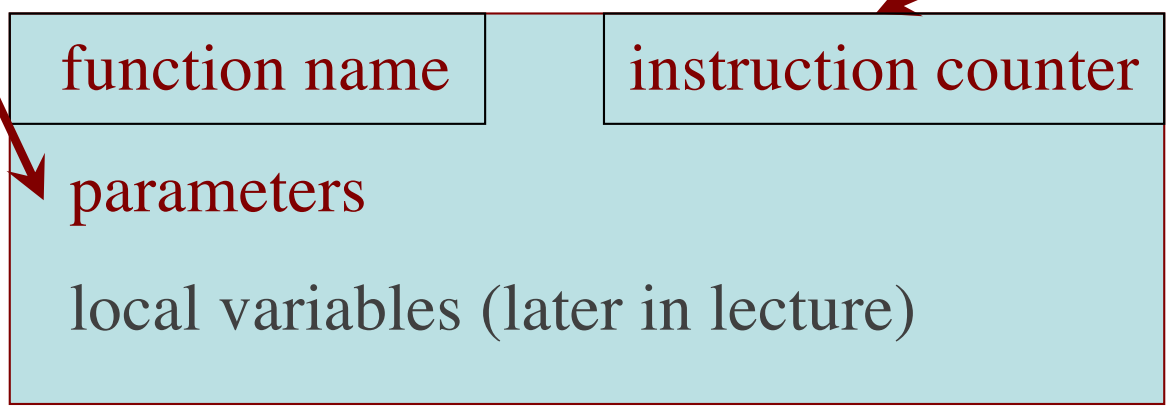
C:/> python simple_math.py

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**



Note: slightly different than in the book (3.9) Please do it **this** way.

Example: get_feet **in** height.py **module**

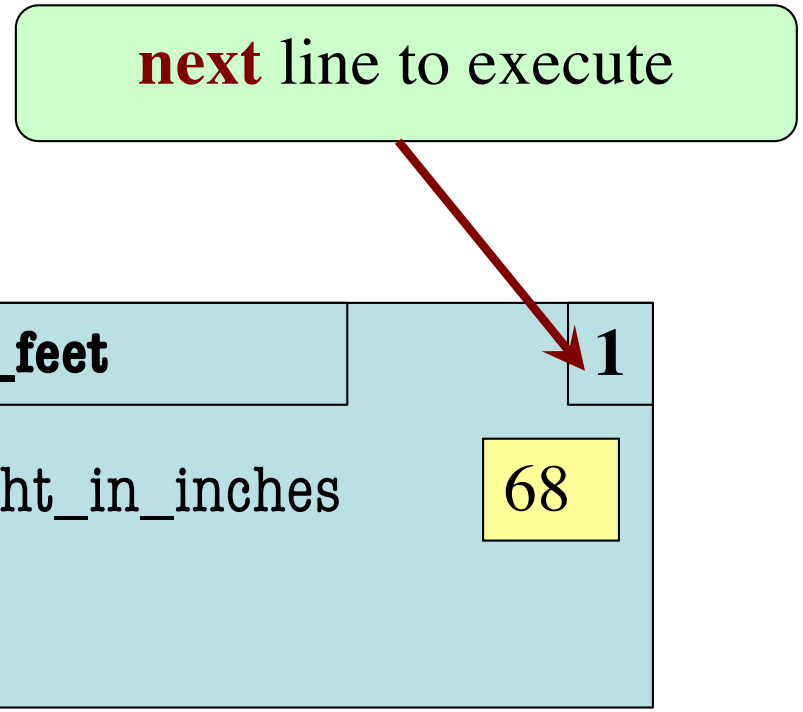
```
>>> import height
>>> height.get_feet(68)
```

```
def get_feet(ht_in_inches):
    return ht_in_inches // 12
```

Example: get_feet(68)

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



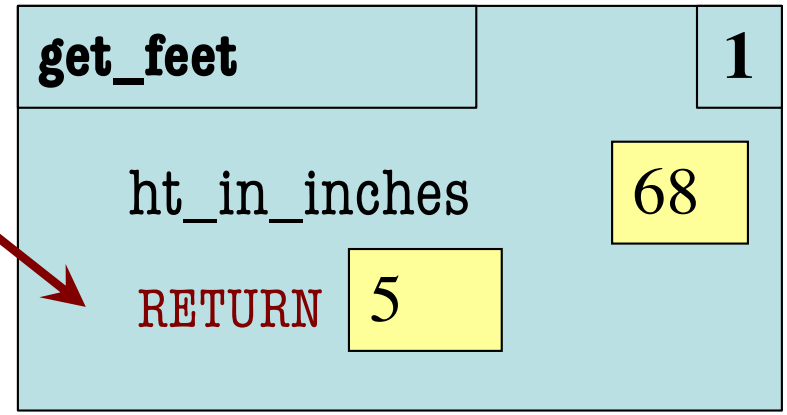

```
def get_feet(ht_in_inches):
    return ht_in_inches // 12
```

Example: get_feet(68)

PHASE 2:

Execute function body

Return statement creates a special variable for result

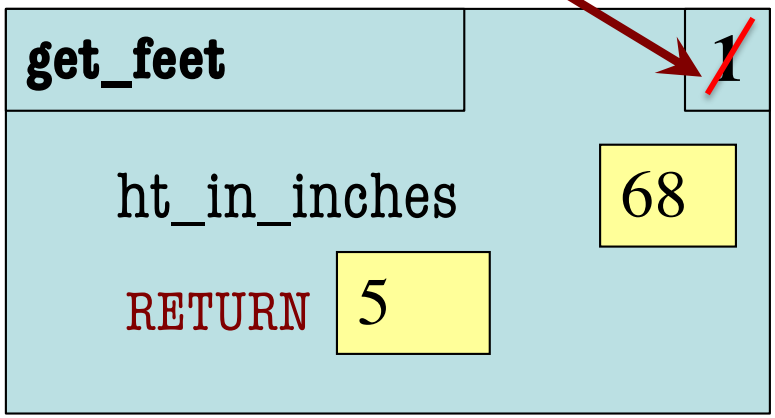



```
def get_feet(ht_in_inches):
    return ht_in_inches // 12
```

Example: get_feet(68)

PHASE 2:
Execute function body

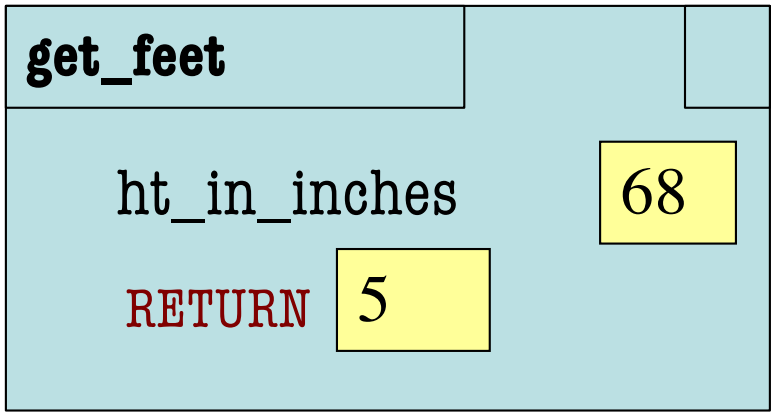
The return terminates;
no next line to execute




```
def get_feet(ht_in_inches):
    return ht_in_inches // 12
```

Example: get_feet(68)

PHASE 3: Erase call frame




```
def get_feet(ht_in_inches):
    return ht_in_inches // 12
```

Example: get_feet(68)

PHASE 3: Erase call frame

But don't actually erase on an exam

ERASE WHOLE FRAME

```
def get_feet(ht_in_inches):
    return ht_in_inches // 12
```

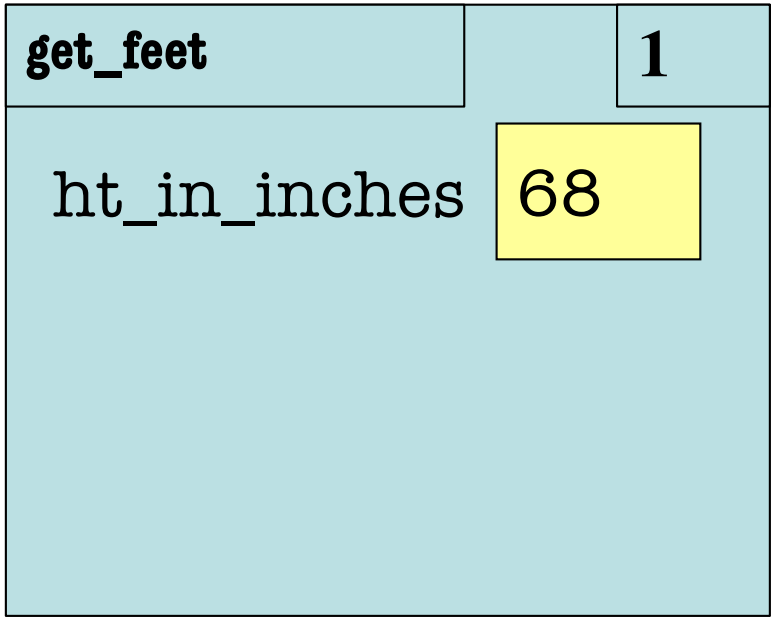
Local Variables (1)

- Call frames can make "local" variables

```
>>> import height
>>> height.get_feet(68)
```

```
def get_feet(ht_in_inches):
    feet = ht_in_inches // 12
    return feet
```

1
2



Local Variables (2)

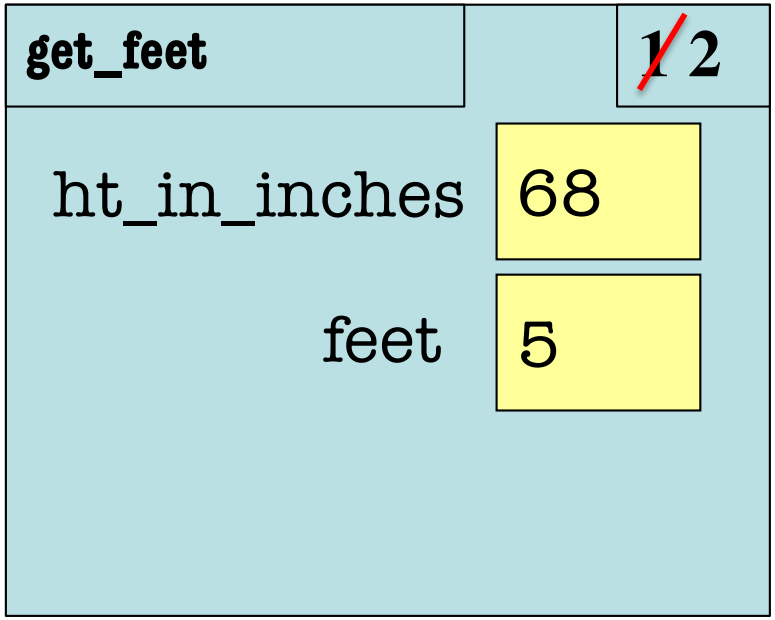
- Call frames can make "local" variables

```
>>> import height
>>> height.get_feet(68)
```

```
def get_feet(ht_in_inches):
    feet = ht_in_inches // 12
    return feet
```

1

2

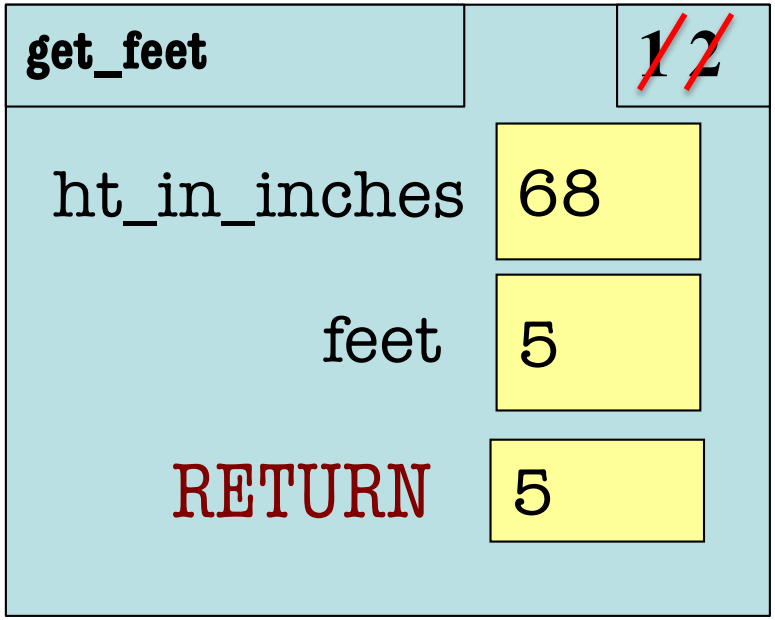


Local Variables (3)

- Call frames can make "local" variables

```
>>> import height
>>> height.get_feet(68)
```

```
def get_feet(ht_in_inches):
    feet = ht_in_inches // 12
    return feet
```



Local Variables (4)

- Call frames can make "local" variables

```
>>> import height
>>> height.get_feet(68)
```

```
def get_feet(ht_in_inches):
    feet = ht_in_inches // 12
    return feet
```

ERASE WHOLE FRAME

Variables are gone! This function is over.

Exercise Time

Function Definition

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

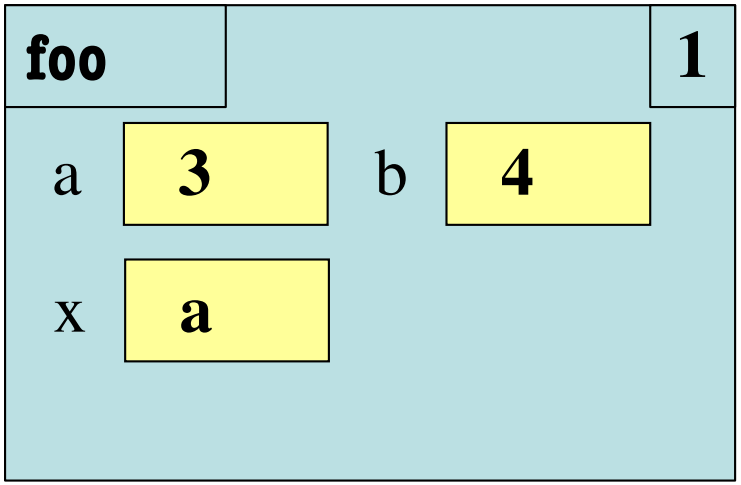
Function Call

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

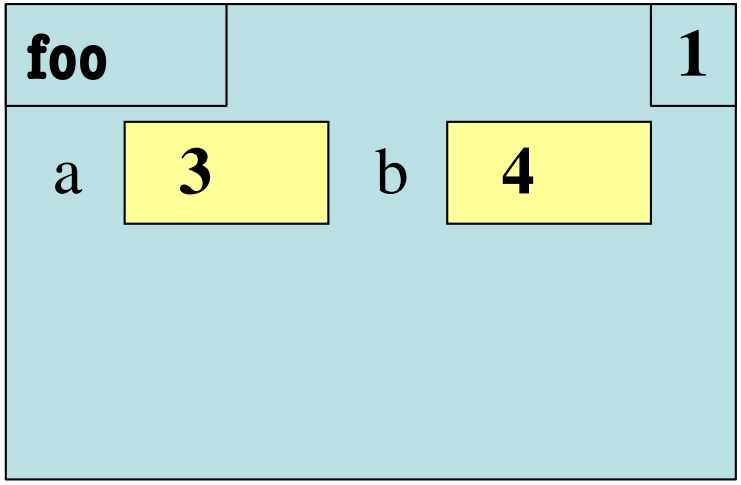
What does the frame look like at the **start**?

Which One is Closest to Your Answer?

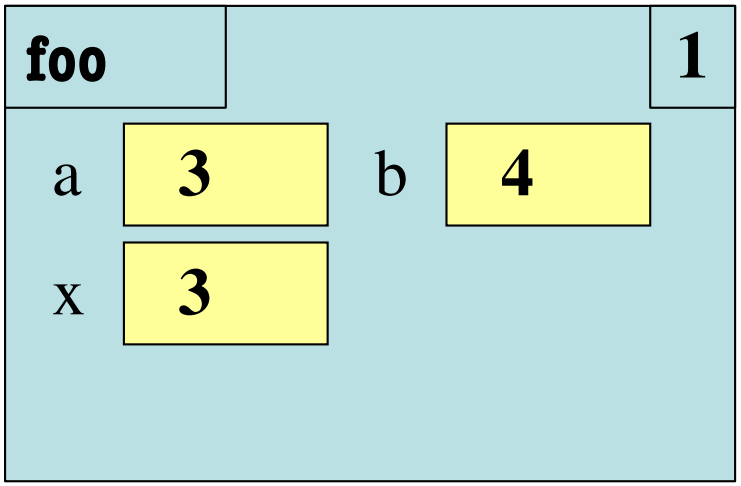
A:



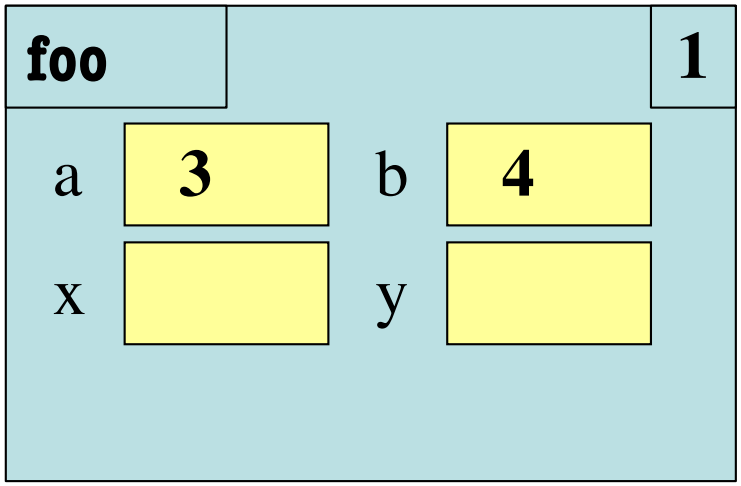
B:



C:

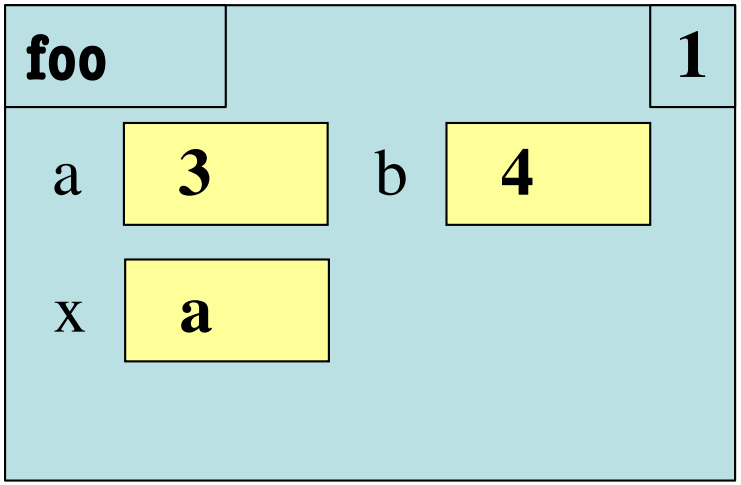


D:

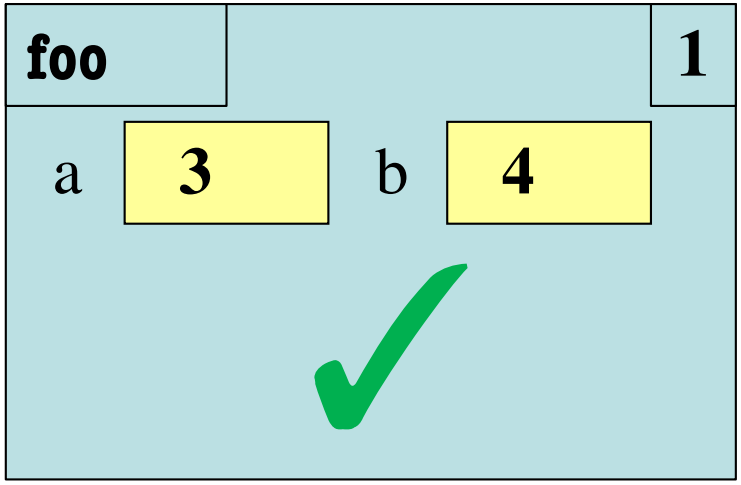


And the answer is…

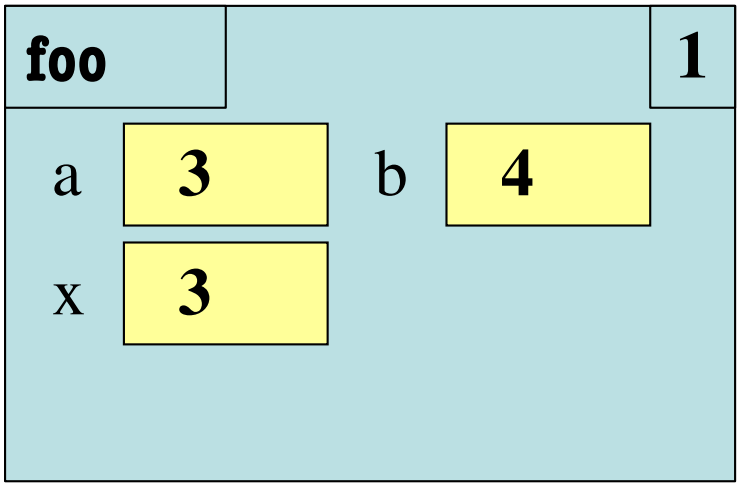
A:



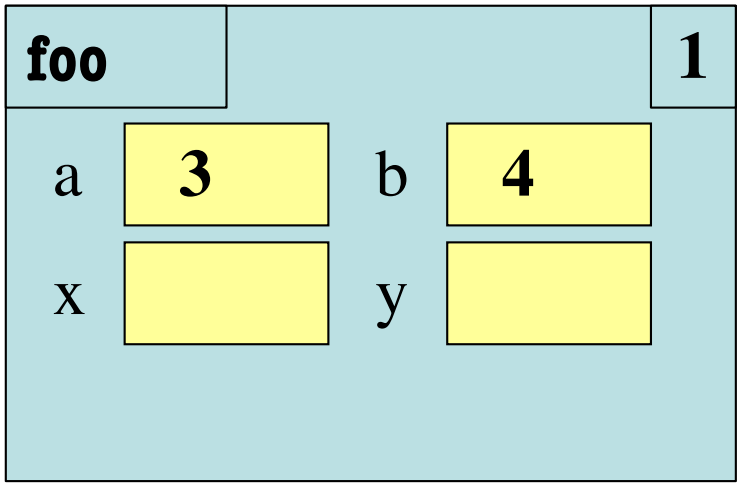
B:



C:



D:



Exercise Time

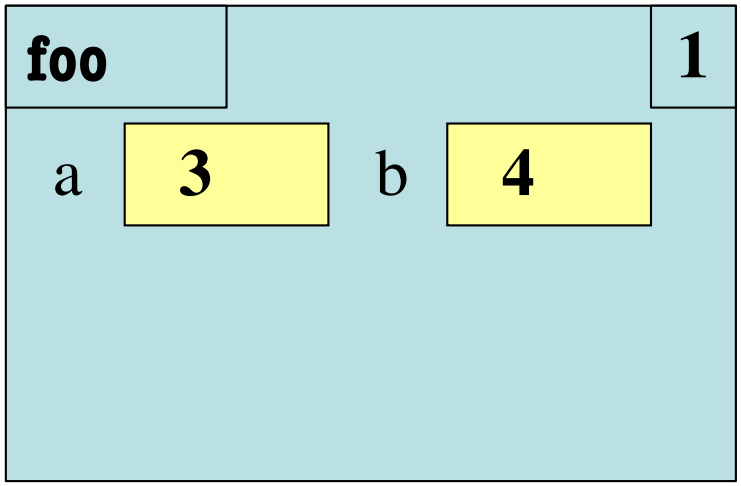
Function Definition

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

Function Call

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

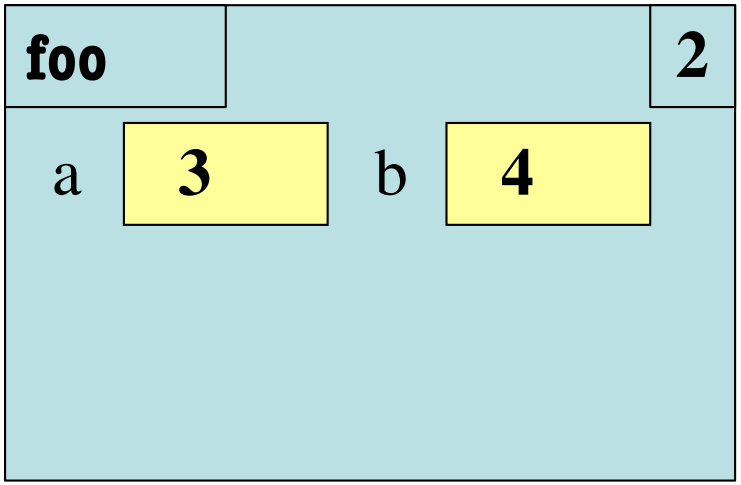
B:



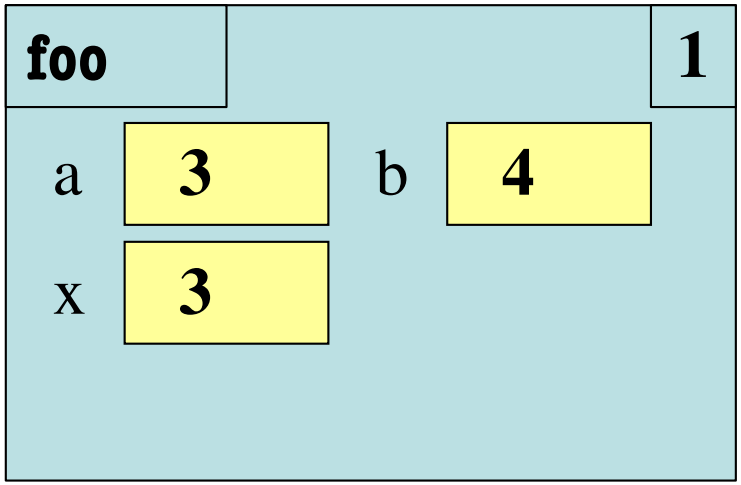
What is the **next step**?

Which One is Closest to Your Answer?

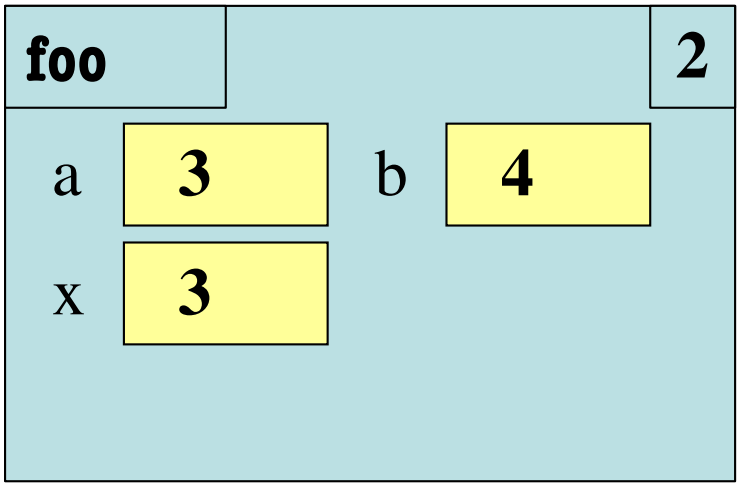
A:



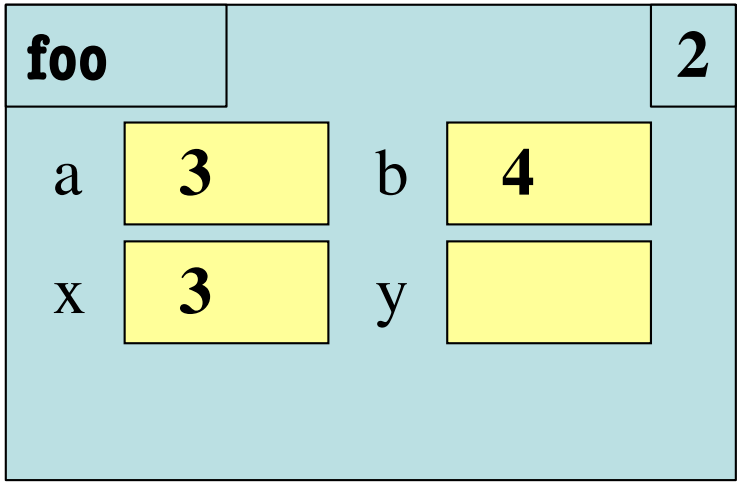
B:



C:

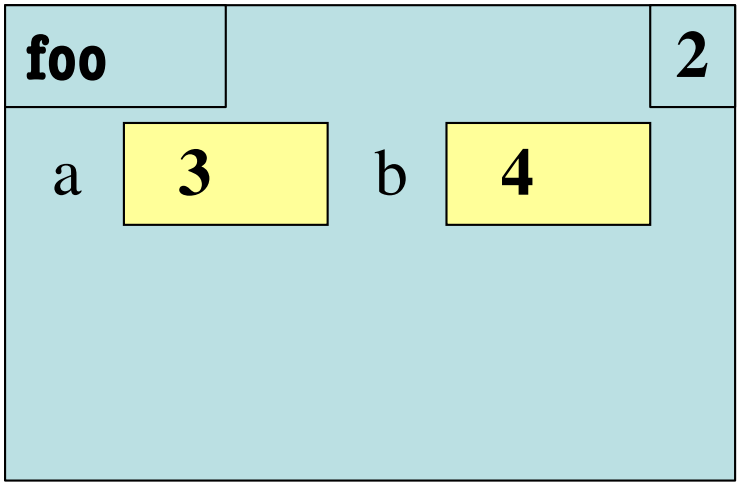


D:

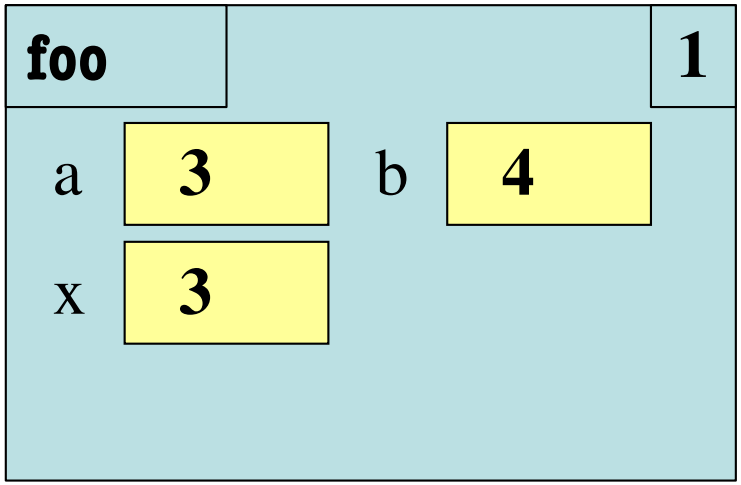


And the answer is…

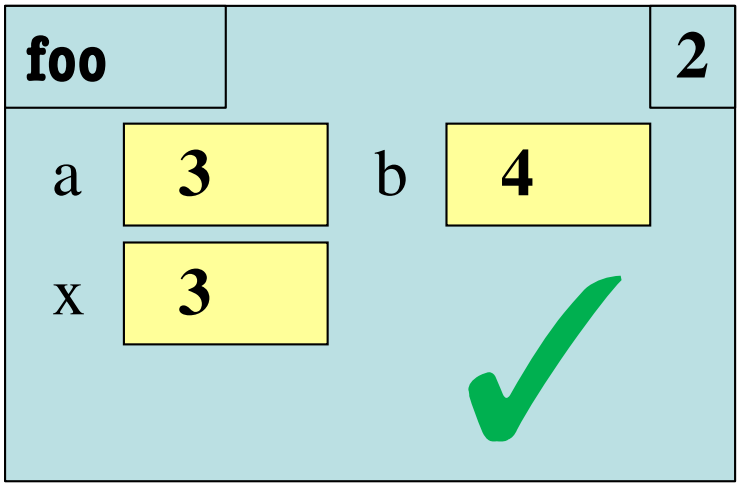
A:



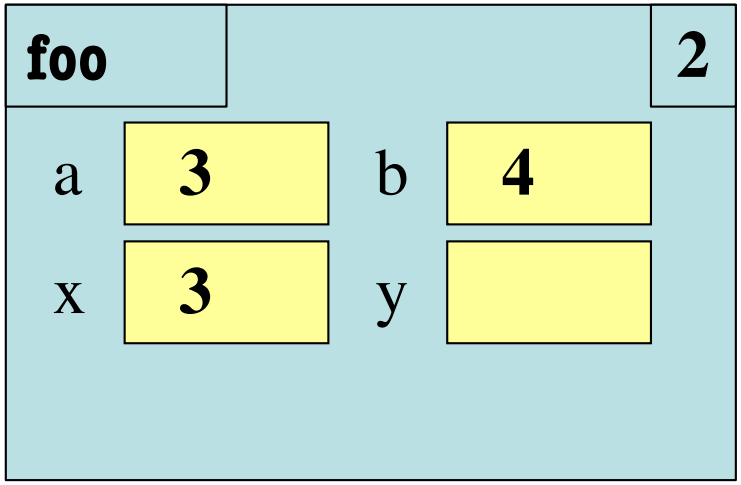
B:



C:



D:



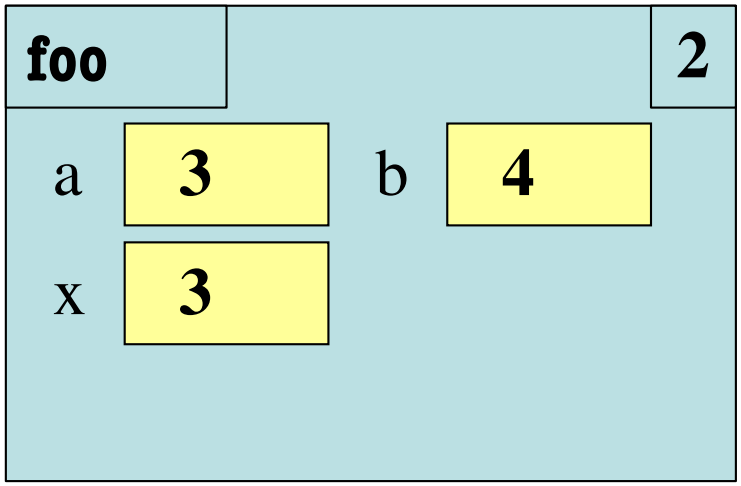
Exercise Time

Function Definition

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

Function Call

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



What is the **next step**?

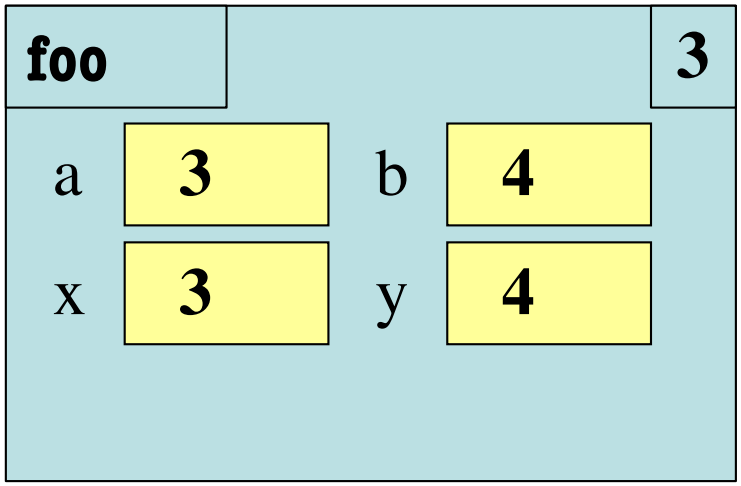
Exercise Time

Function Definition

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

Function Call

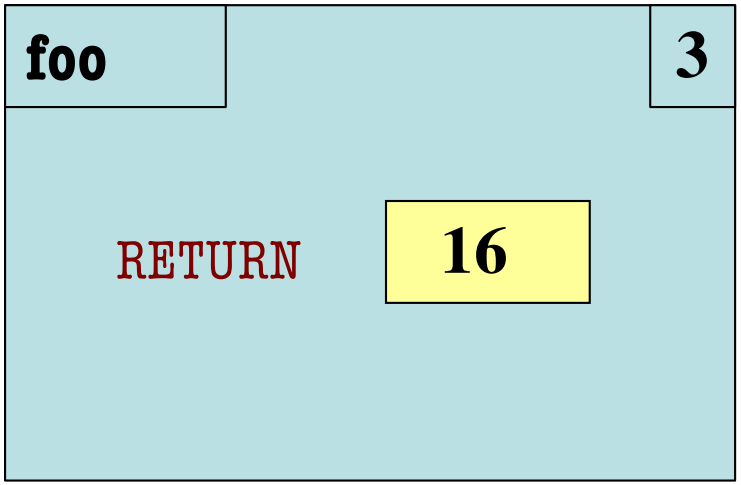
```
>>> foo(3,4)
```



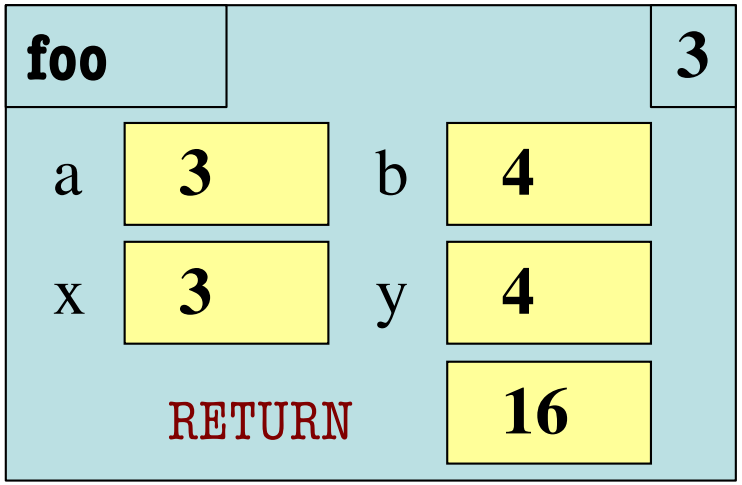
What is the **next step**?

Which One is Closest to Your Answer?

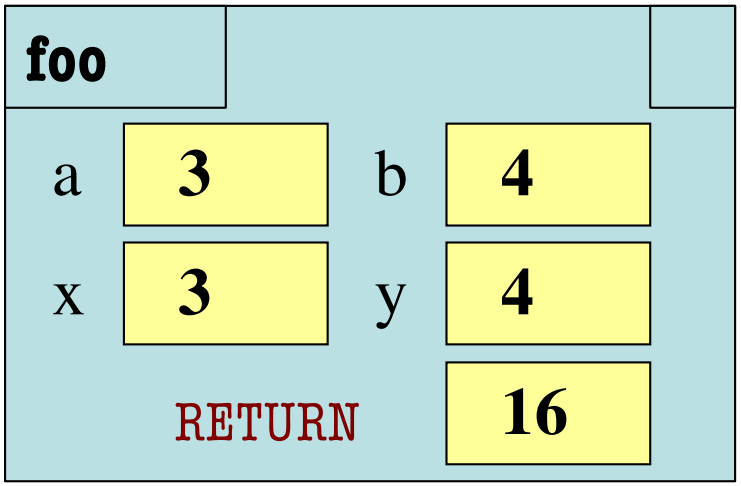
A:



B:



C:

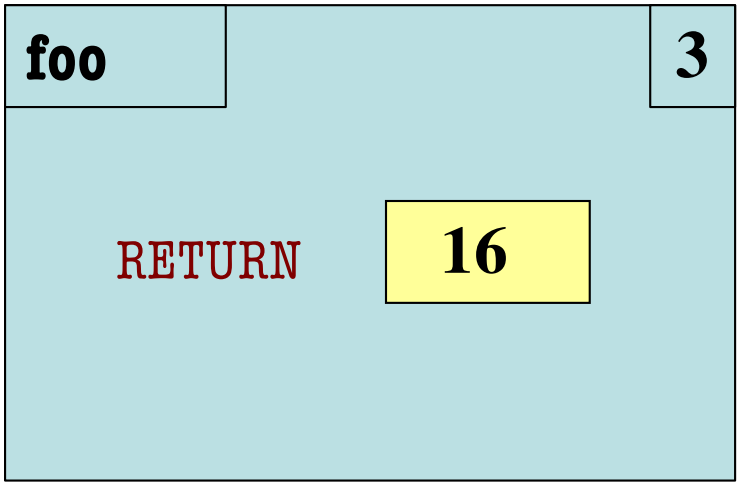


D:

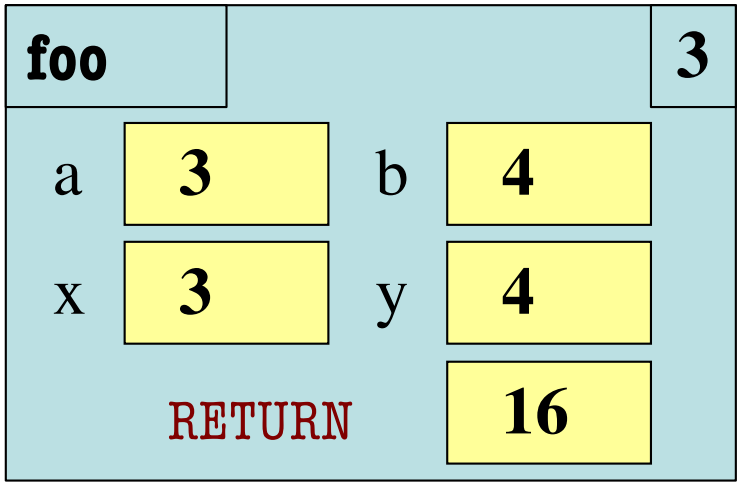
ERASE THE FRAME

And the answer is…

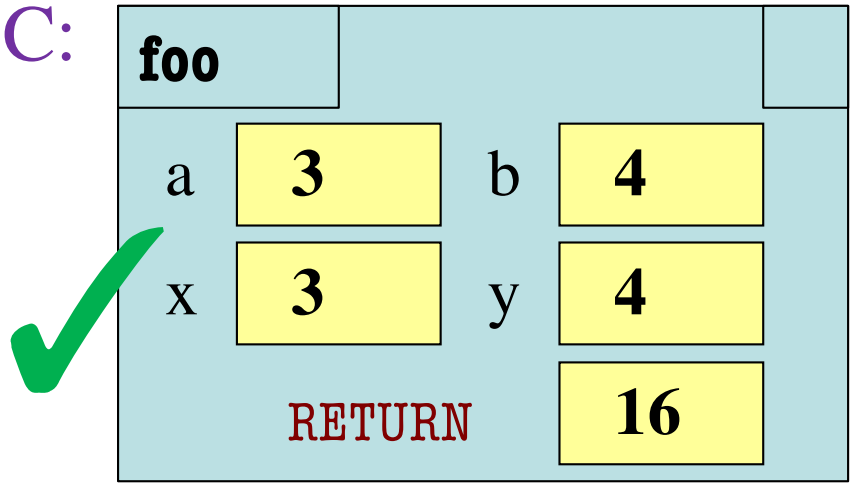
A:



B:



C:



D:

ERASE THE FRAME

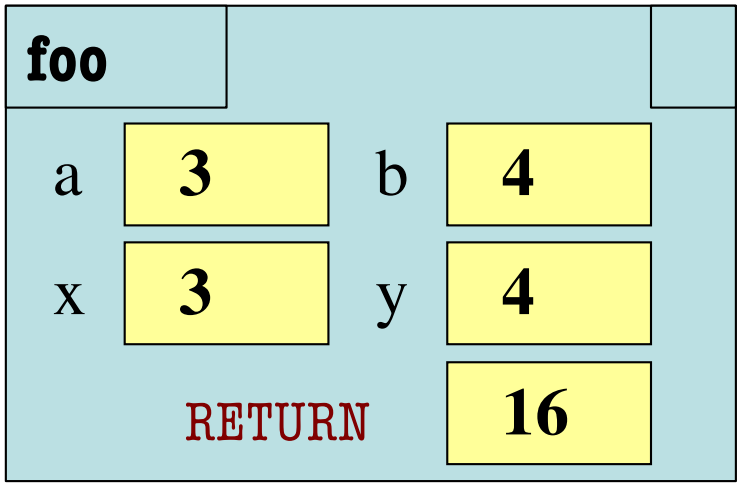
Exercise Time

Function Definition

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

Function Call

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



What is the **next step**?

Exercise Time

Function Definition

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

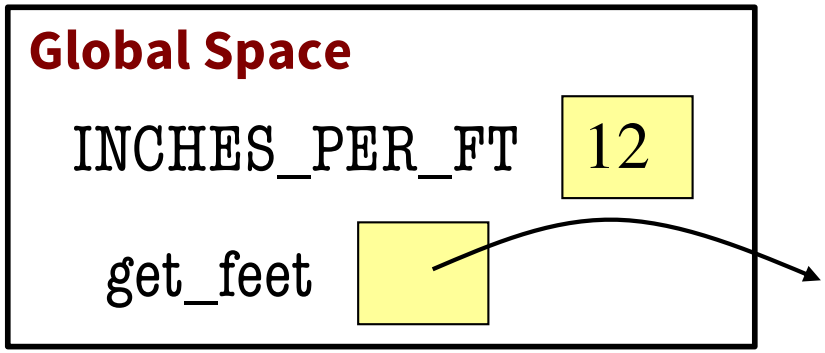
Function Call

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

ERASE THE FRAME

Function Access to Global Space

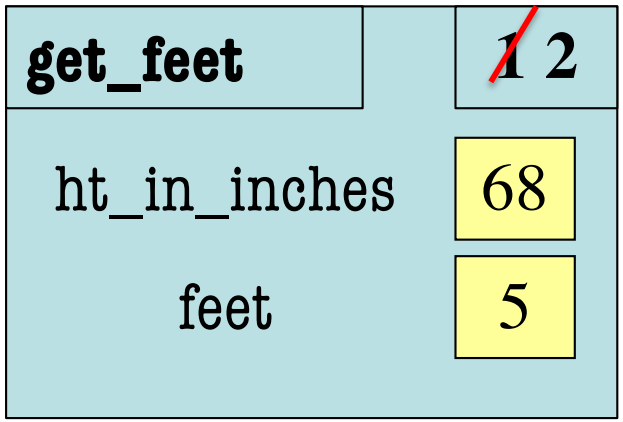
- Top-most location in memory called global space
- Functions can access anything in that global space




```
INCHES_PER_FT = 12

...

def get_feet(ht_in_inches):

    feet = ht_in_inches // INCHES_PER_FT
    return feet

get_feet(68)
```

1

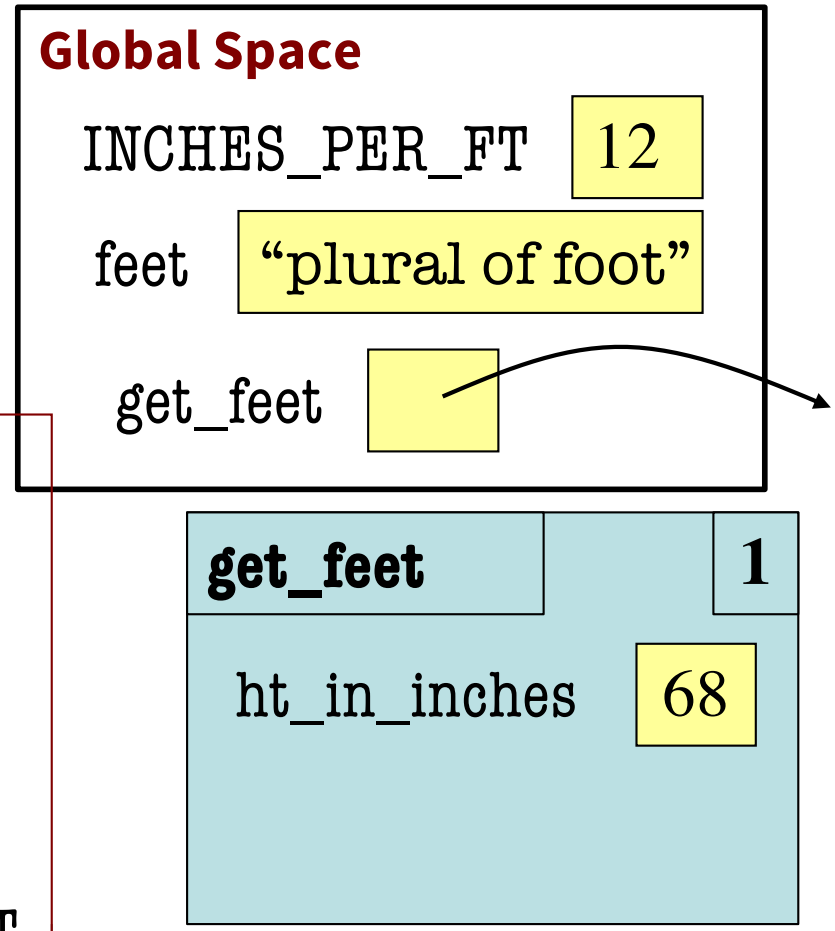
2



What about this??

- What if you choose a local variable inside a function that happens to also be a global variable?

```
INCHES_PER_FT = 12

feet = "plural of foot"

...

def get_feet(ht_in_inches):
1    feet = ht_in_inches // INCHES_PER_FT
2    return feet

get_feet(68)
```



Look, but don't touch!

Can't change global variables

"Assignment to a global" makes a new local variable!

```
INCHES_PER_FT = 12

feet = "plural of foot"

...

def get_feet(ht_in_inches):

    feet = ht_in_inches // INCHES_PER_FT
    return feet

get_feet(68)
```

1

2