Lecture 14

Recursion

Announcements for Today

Assignments

- Assignment 3 now graded
  - **Mean** 96.3, **Median** 99
  - **Time**: 7.5 hr, **StdDev**: 3.5 hr
  - With 666 responses (nice!)
- Assignment 4 is now up!
  - Parts 1-3: Can do already
  - Part 4: Material from today
  - Part 5: Covered on Thursday
  - Due in two weeks

Other Announcements

- View the lesson videos
  - **Videos 17.1-16.5** for today
  - **Videos 17.6-17.11** next time
  - New videos posted Thursday
- Prelim to be graded **Saturday**
  - Will post grade in evening
  - Will give grade boundaries
  - In time for drop deadline
  - **But Bs are good grades!**

Activity Time: The Call Stack

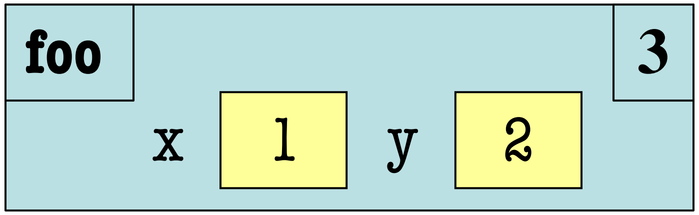
Function Definitions

```
1 def foo(x):
2     y = x+1
3     return bar(y+1)
4
5 def bar(y):
6     return foo(y-1)
```

Function Call

```
>>> foo(1)
```

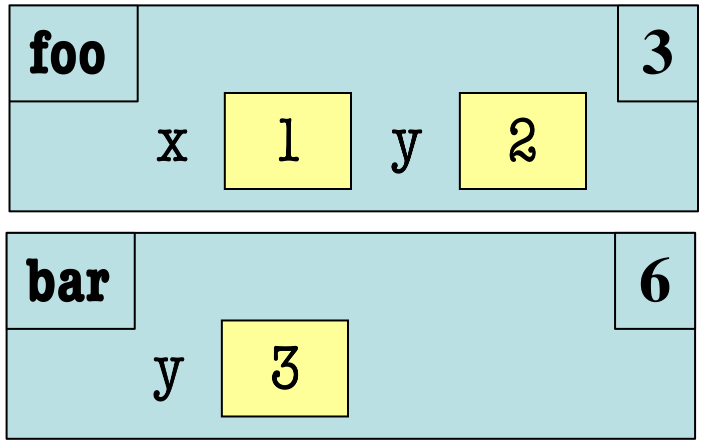
Assume we are here:



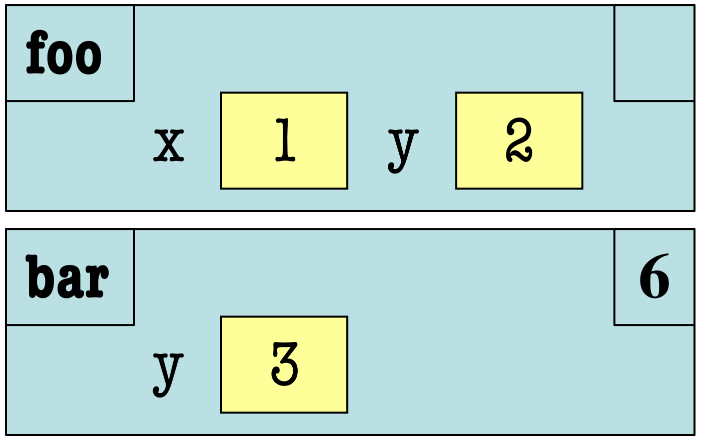
What is the **next step**?

Which One is Closest to Your Answer?

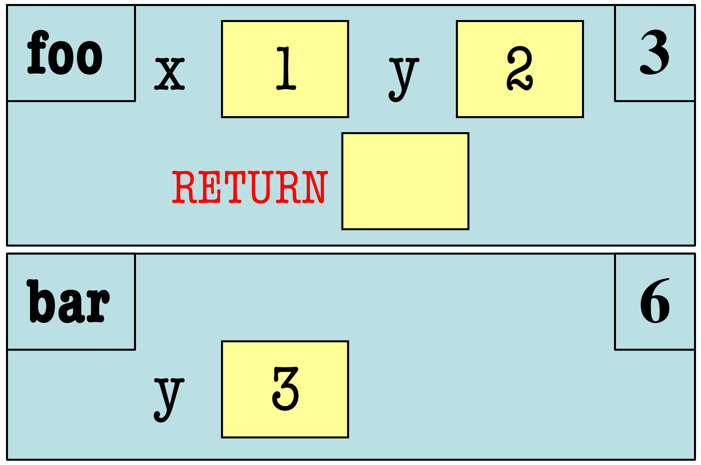
A:



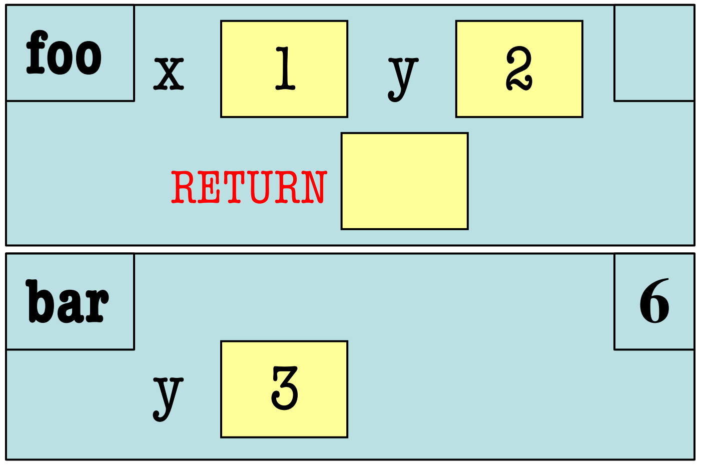
B:



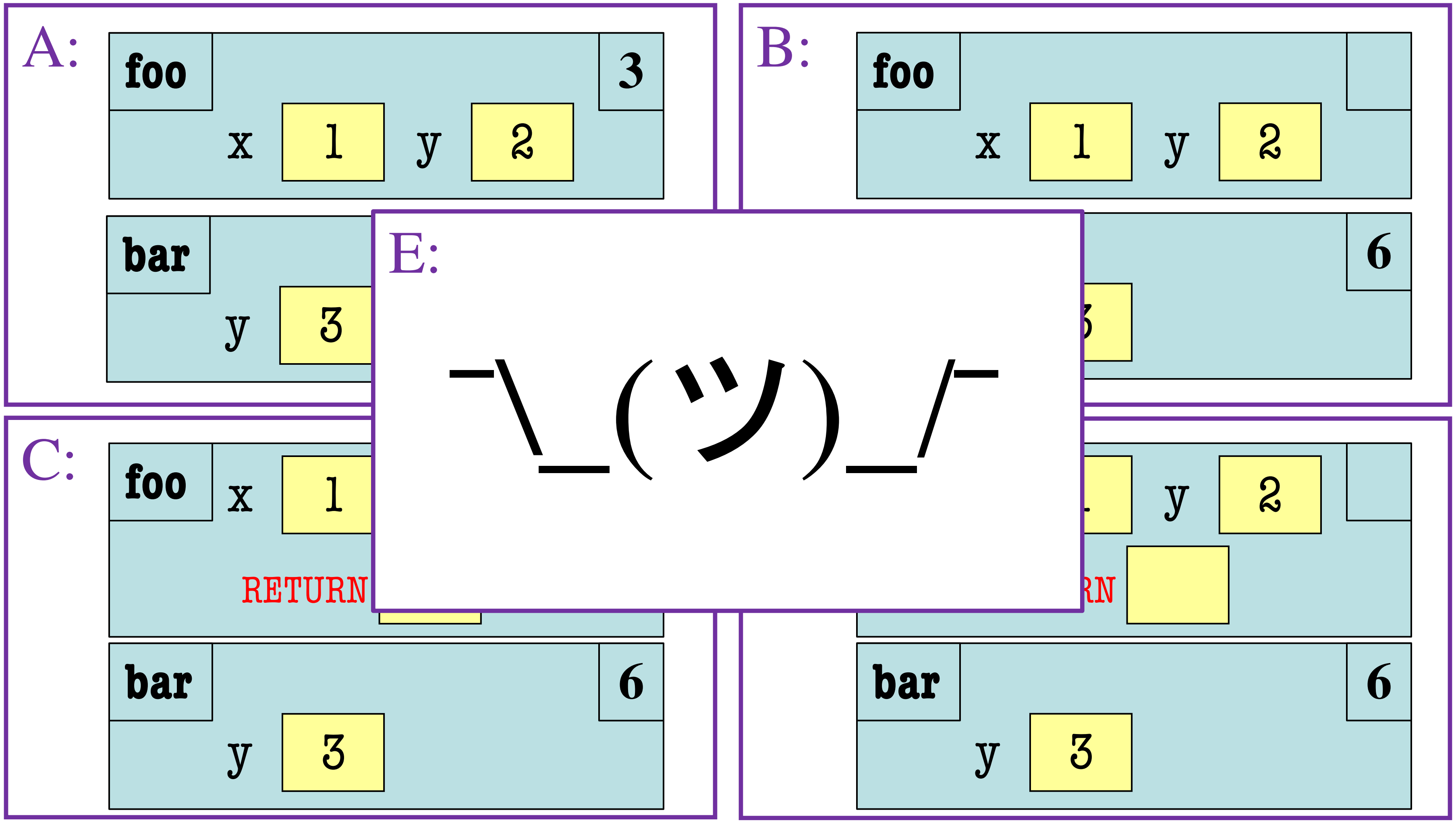
C:



D:



Which One is Closest to Your Answer?



Activity Time: The Call Stack

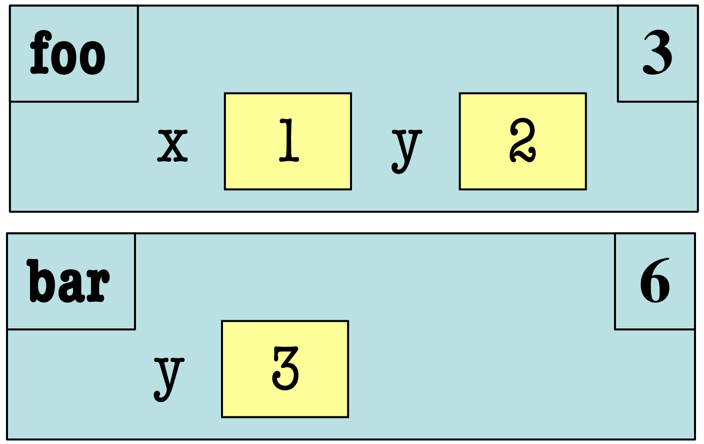
Function Definitions

```
1 def foo(x):
2     y = x+1
3     return bar(y+1)
4
5 def bar(y):
6     return foo(y-1)
```

Function Call

```
>>> foo(1)
```

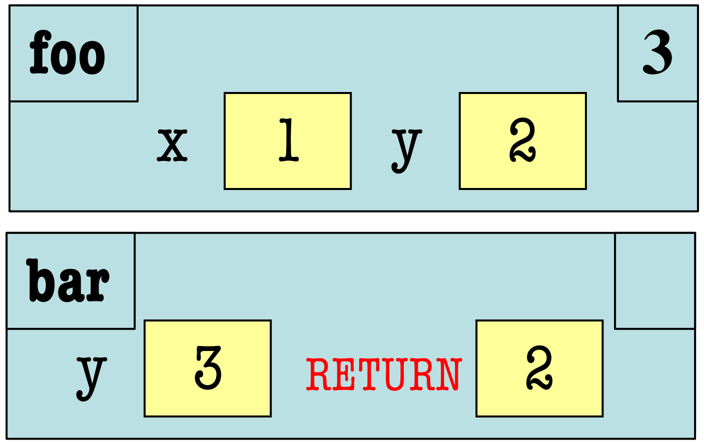
A:



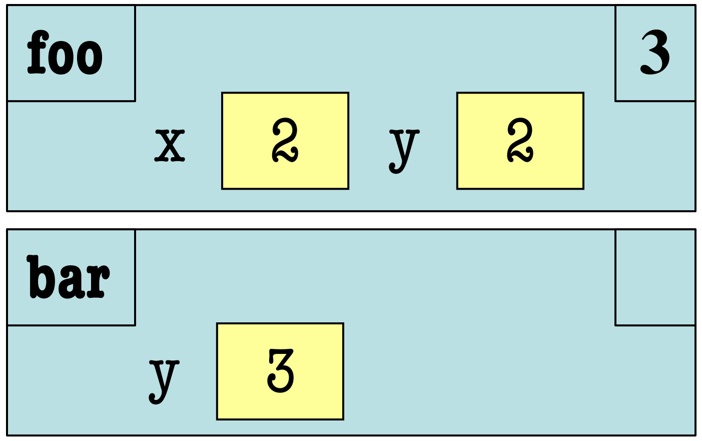
What is the **next step**?

Which One is Closest to Your Answer?

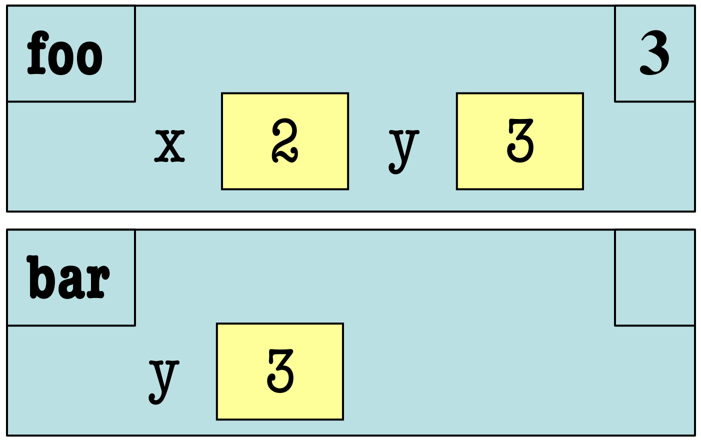
A:



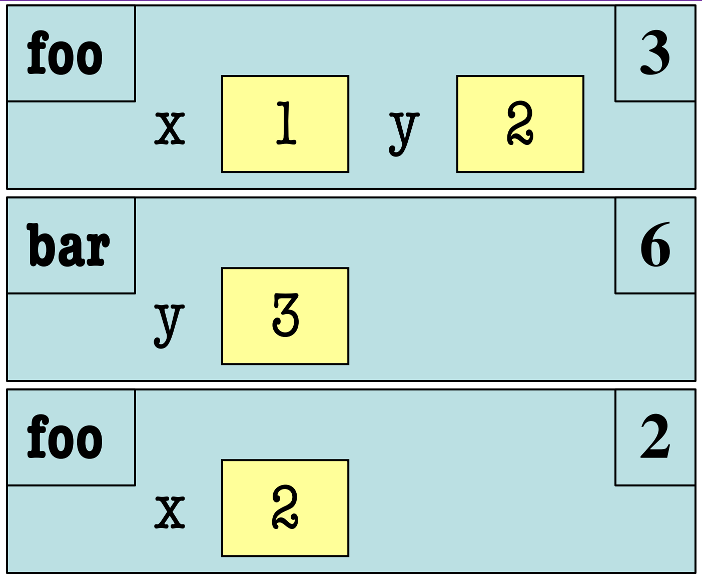
B:



C:



D:



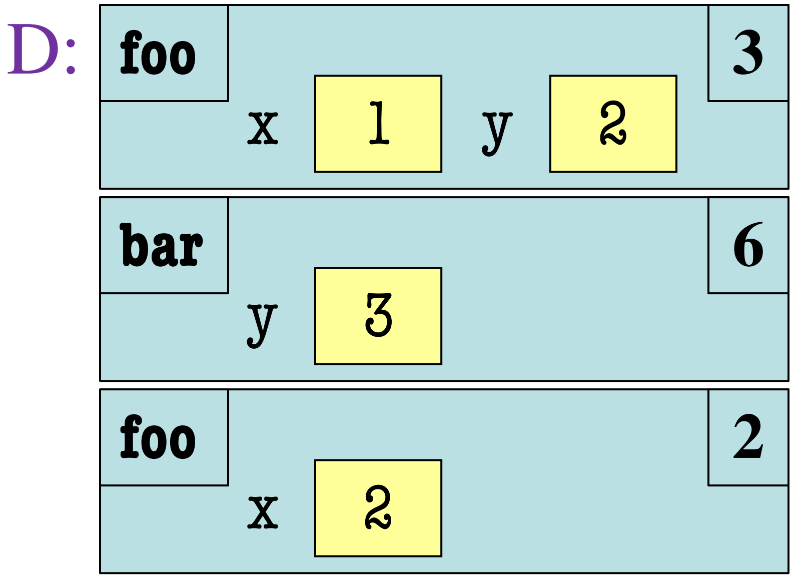
Activity Time: The Call Stack

Function Definitions

```
1 def foo(x):
2     y = x+1
3     return bar(y+1)
4
5 def bar(y):
6     return foo(y-1)
```

Function Call

```
>>> foo(1)
```



This is called
Mutal Recursion

Simple Recursive Function

```
def lucas(n,p,q):
    """
    Returns the nth Lucas number for coefficients p and q.

    A Lucas number is a generalization of the Fibonacci Sequence.
    The nth Lucas number L(n) is given by the recursive definition

        L(0) = 0
        L(1) = 1
        L(n) = p*L(n-1) - q*L(n-2)

    Preconditions: n is an int >= 0, p and q are ints
    """
    pass
```

Simple Recursive Function

```
def lucas(n,p,q):
    """
    Returns the nth Lucas number for coef

    A Lucas number is a generalization of t
    The nth Lucas number L(n) is given by

        L(0) = 0
        L(1) = 1
        L(n) = p*L(n-1) - q*L(n-2)

    Preconditions: n is an int >= 0, p is an
    """
    pass
```

Base Case?

A: n = 0

B: n = 1

C: n = 0, n = 1

D: n = 0, p = 0

E: n = 0, p = 0, q = 0

Simple Recursive Function

```
def lucas(n,p,q):
    """
    Returns the nth Lucas number for coef

    A Lucas number is a generalization of t
    The nth Lucas number L(n) is given by

        L(0) = 0
        L(1) = 1
        L(n) = p*L(n-1) - q*L(n-2)

    Preconditions: n is an int >= 0, p is an
    """
    pass
```

Base Case?

A: n = 0

B: n = 1

[✓ C: n = 0, n = 1]

D: n = 0, p = 0

E: n = 0, p = 0, q = 0

Divide and Conquer

```
def prod(tup):
    """
    Returns the product of the integers in tup. Returns 1 if empty.

    Examples:
        prod((12,)) returns 12
        prod((7,12,1,2,2)) returns 336
        prod(()) returns 1

    Precondition: tup is a tuple of ints
    """
    pass
```

Divide and Conquer

```
def prod(tup):
    """
    Returns the product of the integers in tup. Returns 1 if empty.

    Examples:
        prod((12,)) returns 12
        prod((7,12,1,2,2)) returns 336
        prod(()) returns 1

    Precondition: tup is a tuple of ints
    """
    pass
```

How Divide?

A: Cut in half

B: Pull off one elt.

C: Does not matter

Divide and Conquer

```
def prod(tup):
    """
    Returns the product of the integers in tup. Returns 1 if empty.

    Examples:
        prod((12,)) returns 12
        prod((7,12,1,2,2)) returns 336
        prod(()) returns 1

    Precondition: tup is a tuple of ints
    """
    pass
```

How Combine?

A: Add left, right

B: Multiply left, right

C: Does not matter

Divide and Conquer 2

```
def depunct(s):
    """
    Returns s but with everything that is not a letter removed

    Examples:
        depunct('Hello') returns 'Hello'
        depunct('Hello World!') returns 'HelloWorld'

    Parameter: s the string to edit
    Precondition s is a string
    """
    pass
```

Divide and Conquer 2

```
def depunct(s):
    """
    Returns s but with everything that is not a letter removed

    Examples:
        depunct('Hello') returns 'Hello'
        depunct('Hello World!') returns 'H
    
    Parameter: s the string to edit
    Precondition s is a string
    """
    pass
```

How Divide?

A: Cut in half

B: Pull off one elt.

C: Does not matter

Divide and Conquer 2

```
def depunct(s):
    """
    Returns s but with everything that is not a letter removed

    Examples:
        depunct('Hello') returns 'Hello'
        depunct('Hello World!') returns 'H

    Parameter: s the string to edit
    Precondition s is a string
    """
    pass
```

How Combine?

A: Add left, right

B: Add right, left

C: Does not matter

Divide and Conquer 3

```
def reverse(s):
    """
    Returns s with its characters in reverse order

    Examples:
        depunct('Hello') returns 'olleH'
        depunct('amma') returns 'amma'

    Parameter: s the string to reverse
    Precondition s is a string
    """
    pass
```

Divide and Conquer 3

```
def reverse(s):
    """
    Returns s with its characters in reverse order

    Examples:
        depunct('Hello') returns 'olleH'
        depunct('amma') returns 'amma'

    Parameter: s the string to reverse
    Precondition s is a string
    """
    pass
```

How Divide?

A: Cut in half

B: Pull off one elt.

C: Does not matter

Divide and Conquer 3

```
def reverse(s):
    """
    Returns s with its characters in reverse order

    Examples:
        depunct('Hello') returns 'olleH'
        depunct('amma') returns 'amma'

    Parameter: s the string to reverse
    Precondition s is a string
    """
    pass
```

How Combine?

A: Add left, right

B: Add right, left

C: Does not matter

Questions?