CS 277: Lecture 2

Reminders

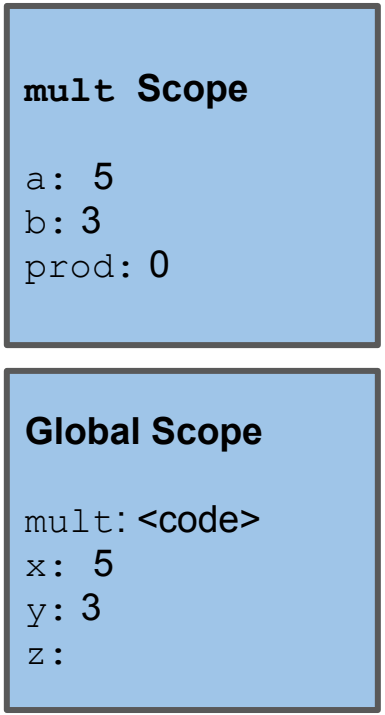
- Class Discussion Forum: Please join Ed Stem ASAP!
- PrairieLearn: Please join the Spring 2026 CS 277 class
- Lab 1 is due on Thursday (on PrairieLearn)
- Homework 1 will be released today (on PrairieLearn)
    - Due Monday in 2 weeks
    - Will have problems on topics covered this week
    - Can be solved in groups of 3

```
def mult(a,b):
  prod = 0
  while (b > 0):
    prod += a
    b -= 1
  return prod

x = 5
y = 3
z = mult(x,y)
print(z)
```

Global Scope

mult: <code>
x: 5
y: 3
z:

```
def mult(a,b):
  prod = 0
  while (b > 0):
    prod += a
    b -= 1
  return prod

x = 5
y = 3
z = mult(x,y)
print(z)
```



After one iteration of the `while` loop

```
def mult(a,b):
  prod = 0
  while (b > 0):
    prod += a
    b -= 1
  return prod

x = 5
y = 3
z = mult(x,y)
print(z)
```

`mult` Scope

`a`: 5
`b`: 2
`prod`: 5

Global Scope

`mult`: <code>
`x`: 5
`y`: 3
`z`:

After two iterations of the `while` loop

```
def mult(a,b):
  prod = 0
  while (b > 0):
    prod += a
    b -= 1
  return prod

x = 5
y = 3
z = mult(x,y)
print(z)
```

`mult` Scope

`a`: 5
`b`: 1
`prod`: 10

Global Scope

`mult`: <code>
`x`: 5
`y`: 3
`z`:

After three iterations of the `while` loop

```
def mult(a,b):
  prod = 0
  while (b > 0):
    prod += a
    b -= 1
  return prod

x = 5
y = 3
z = mult(x,y)
print(z)
```

`mult` Scope

`a`: 5
`b`: 0
`prod`: 15

Global Scope

`mult`: <code>
`x`: 5
`y`: 3
`z`:

Executing the `return` statement

```
def mult(a,b):
  prod = 0
  while (b > 0):
    prod += a
    b -= 1
  return prod

x = 5
y = 3
z = mult(x,y)
print(z)
```

```
def mult(a,b):
  prod = 0
  while (b > 0):
    prod += a
    b -= 1
  return prod

x = 5
y = 3
z = mult(x,y)
print(z)
```

Global Scope

mult: <code>
x: 5
y: 3
z: 15

Scope

```
def mult(a,b):
    prod = 0
    print(x)
    while (b > 0):
        prod += a
        b -= 1
    return prod

x = 5
y = 3
z = mult(x,y)
print(z)
```

`mult` Scope

`a`: 5
`b`: 3
`prod`: 0

Global Scope

`mult`: <code>
`x`: 5
`y`: 3
`z`:

Scope

```
def mult(a,b):
    prod = 0
    print(x)
    while (b > 0):
        prod += a
        b -= 1
    return prod

x = 5
y = 3
z = mult(x,y)
print(z)
```

Interpret as identifier in the closest enclosing context. In this case it would be the global context.

`mult` Scope

`a`: 5
`b`: 3
`prod`: 0

Global Scope

`mult`: <code>
`x`: 5
`y`: 3
`z`:

Recursion in Action

```
def mult_rec(a,b):
  if b == 0:
    return 0
  return a+mult_rec(a,b-1)

x = 5
y = 3
z = mult_rec(x,y)
print(z)
```

Global Scope
mult_rec: <code>
x: 5
y: 3
z:

Recursion in Action

```
def mult_rec(a,b):
  if b == 0:
    return 0
  return a+mult_rec(a,b-1)

x = 5
y = 3
z = mult_rec(x,y)
print(z)
```

`mult_rec` Scope
`a`: 5
`b`: 3

Global Scope
`mult_rec`: <code>
`x`: 5
`y`: 3
`z`:

Recursion in Action

```
def mult_rec(a,b):
  if b == 0:
    return 0
  return a+mult_rec(a,b-1)

x = 5
y = 3
z = mult_rec(x,y)
print(z)
```

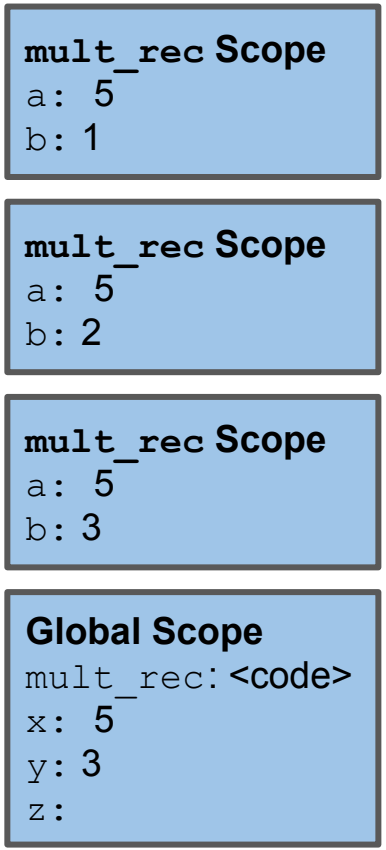
mult_rec Scope
a: 5
b: 2

mult_rec Scope
a: 5
b: 3

Global Scope
mult_rec: <code>
x: 5
y: 3
z:

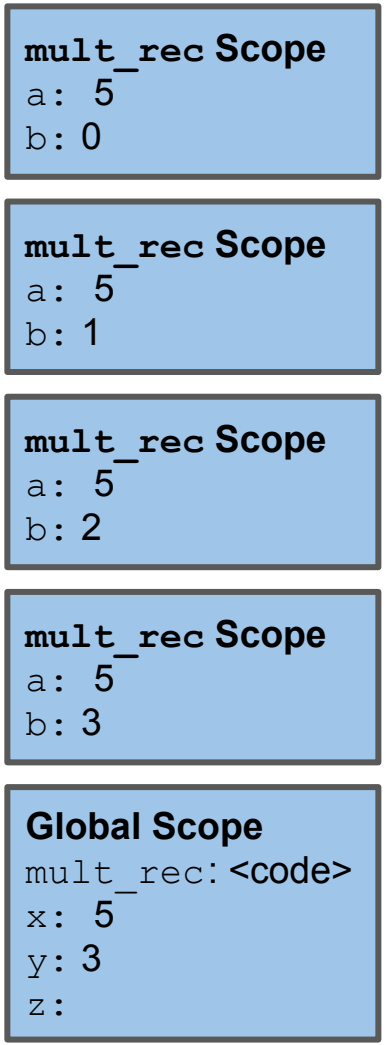
Recursion in Action

```
def mult_rec(a,b):
  if b == 0:
    return 0
  return a+mult_rec(a,b-1)

x = 5
y = 3
z = mult_rec(x,y)
print(z)
```



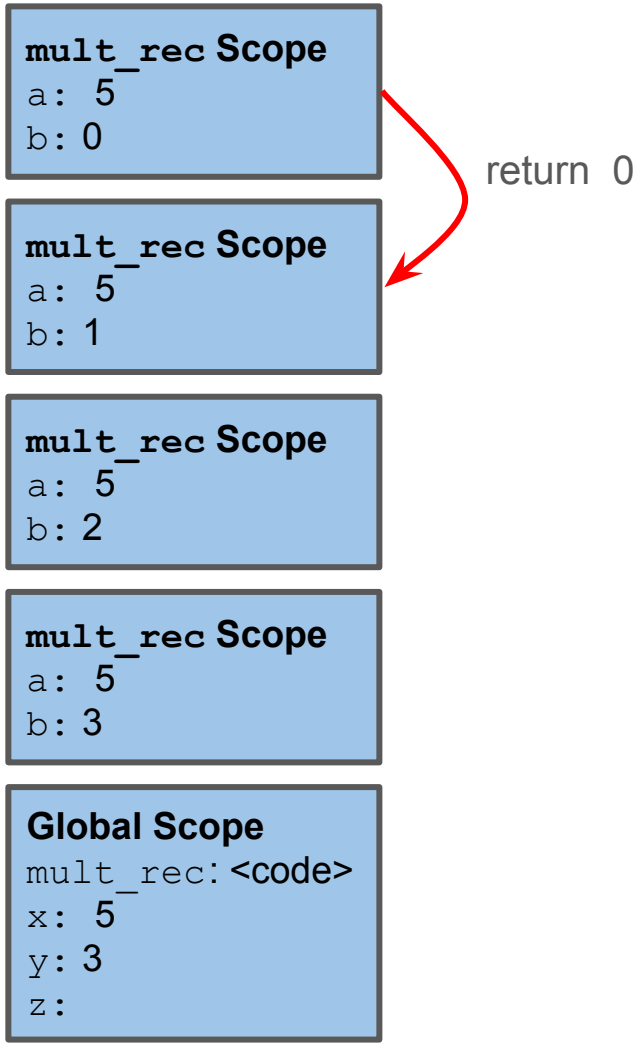
Recursion in Action

```
def mult_rec(a,b):
  if b == 0:
    return 0
  return a+mult_rec(a,b-1)

x = 5
y = 3
z = mult_rec(x,y)
print(z)
```



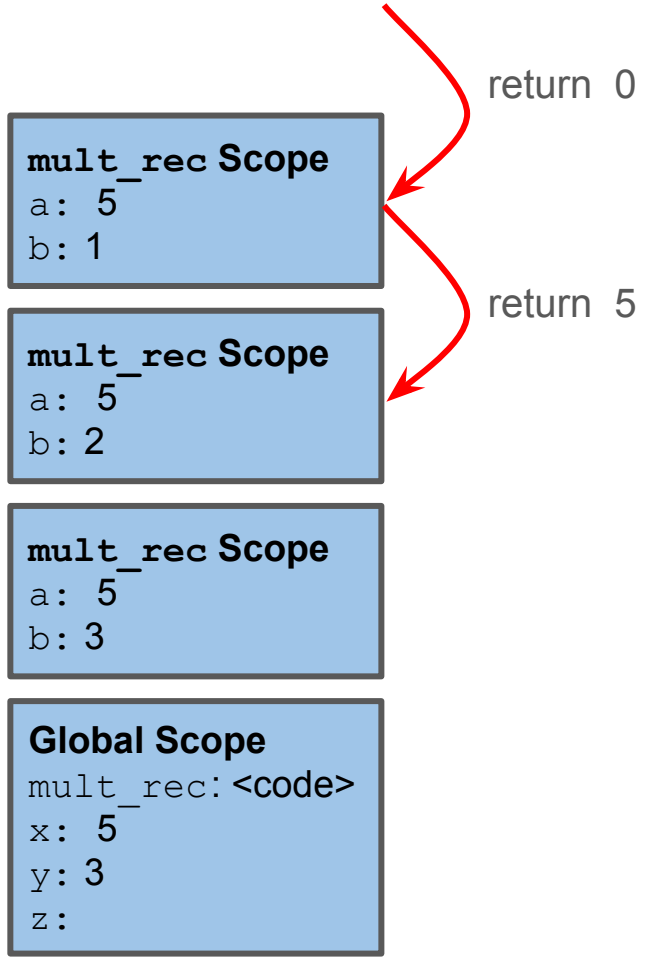
Recursion in Action

```
def mult_rec(a,b):
  if b == 0:
    return 0
  return a+mult_rec(a,b-1)

x = 5
y = 3
z = mult_rec(x,y)
print(z)
```



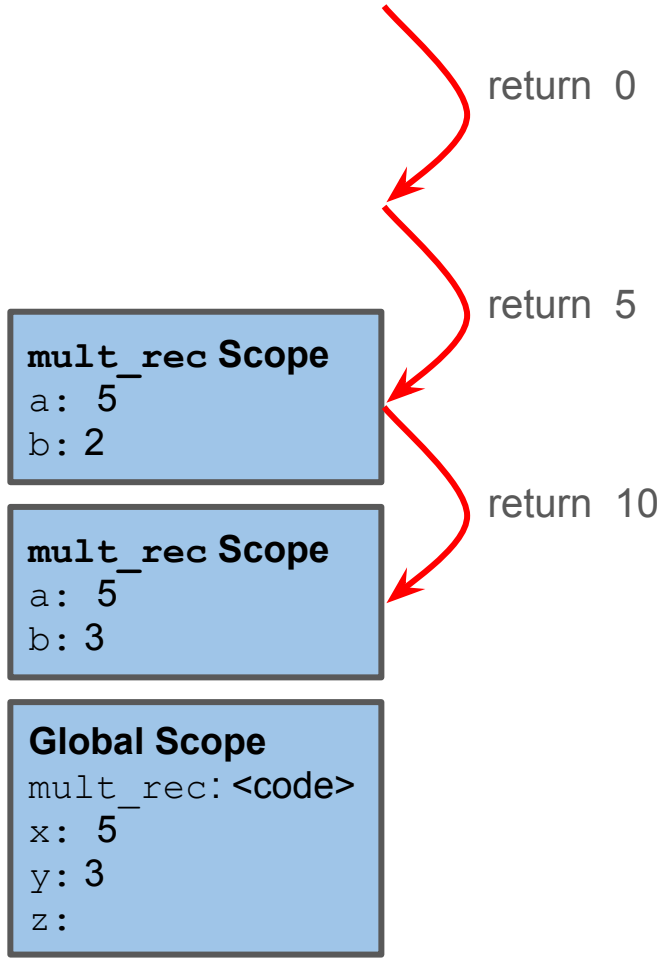
Recursion in Action

```
def mult_rec(a,b):
  if b == 0:
    return 0
  return a+mult_rec(a,b-1)

x = 5
y = 3
z = mult_rec(x,y)
print(z)
```



Recursion in Action

```
def mult_rec(a,b):
  if b == 0:
    return 0
  return a+mult_rec(a,b-1)

x = 5
y = 3
z = mult_rec(x,y)
print(z)
```



Recursion in Action

```
def mult_rec(a,b):
  if b == 0:
    return 0
  return a+mult_rec(a,b-1)

x = 5
y = 3
z = mult_rec(x,y)
print(z)
```