



Lecture 9: Memory in Python

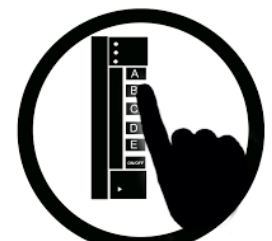
CS 1110

Introduction to Computing Using Python

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

Do you have a conflict with Prelim 1?

- A. Yes, and I've filled out the Prelim 1 Conflict Survey.
- B. Yes, but I haven't yet filled out the Prelim 1 Conflict Survey.
- C. I have this sinking feeling I might have a conflict.
- D. I know I do not have a conflict.
- E. I have no idea.



Global Space

Global Space

- What you “start with”
- Stores global variables
- Lasts until you quit Python

Global Space

x 4

x = 4

Enter Heap Space

Global Space

- What you “start with”
- Stores global variables
- Lasts until you quit Python

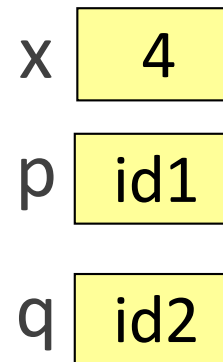
Heap Space

- Where “folders” are stored
- Have to access indirectly

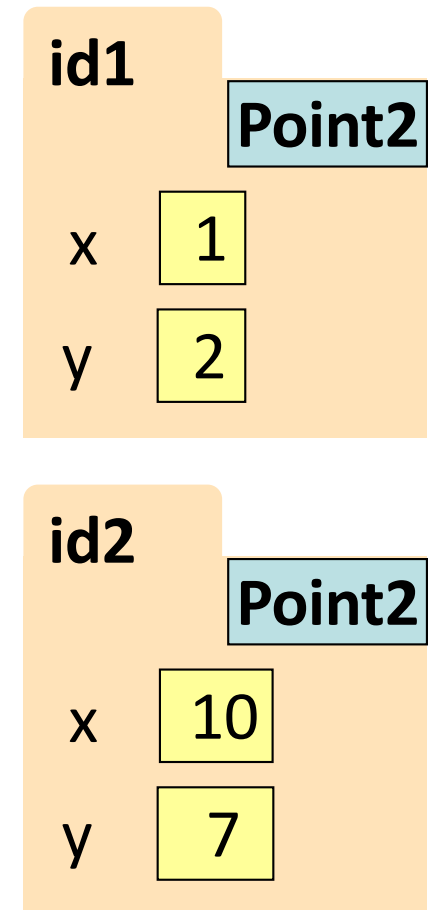
```
x = 4  
p = shape.Point2(1,2)  
q = shape.Point2(10,7)
```

*assumes we
imported shape*

Global Space



Heap Space



p & q live in Global Space. Their folders live on the Heap.

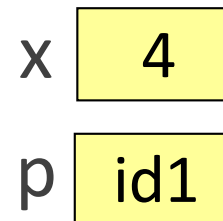
Calling a Function Creates a Call Frame (1)

What's in a Call Frame?

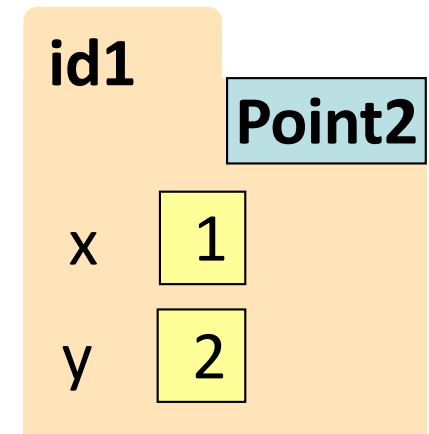
- Boxes for parameters **at the start of the function**
- Boxes for variables local to the function **as they are created**

```
1 def adjust_x(pt, n):  
    pt.x = pt.x + n  
  
    x = 4  
    p = shape.Point2(1,2)  
    adjust_x(p, x)
```

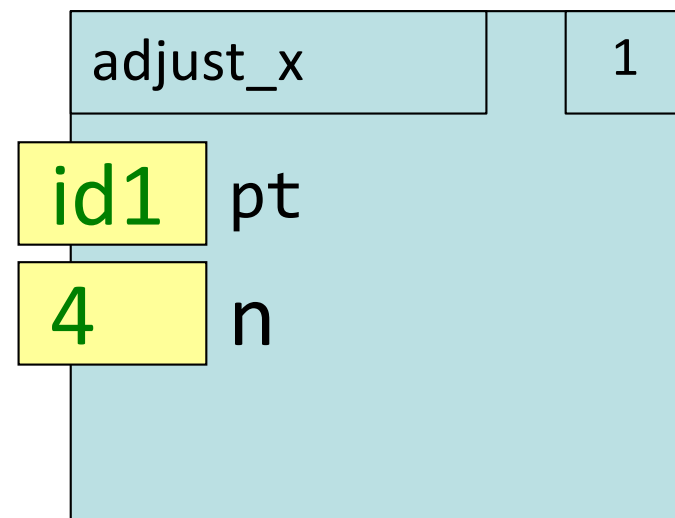
Global Space



Heap Space



Call Stack



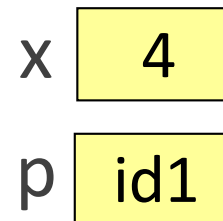
Calling a Function Creates a Call Frame (2)

What's in a Call Frame?

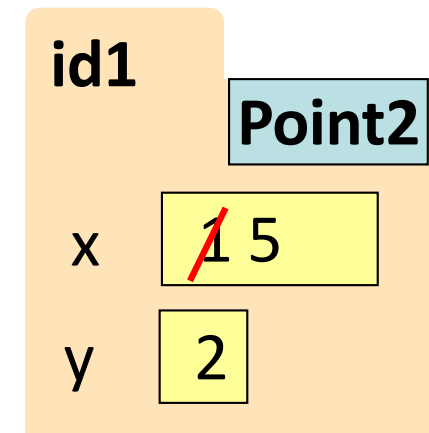
- Boxes for parameters **at the start of the function**
- Boxes for variables local to the function **as they are created**

```
def adjust_x(pt, n):  
1 pt.x = pt.x + n  
  x = 4  
  p = shape.Point2(1,2)  
  adjust_x(p, x)
```

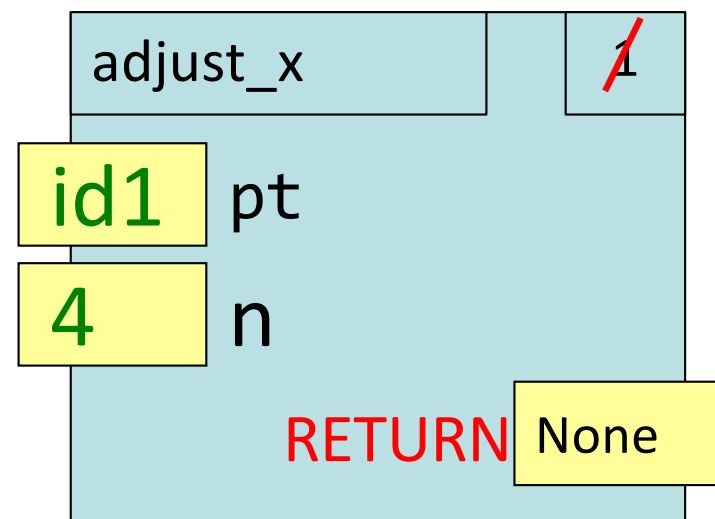
Global Space



Heap Space



Call Stack



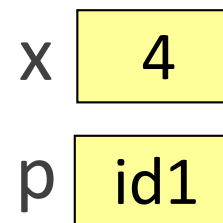
Calling a Function Creates a Call Frame (3)

What's in a Call Frame?

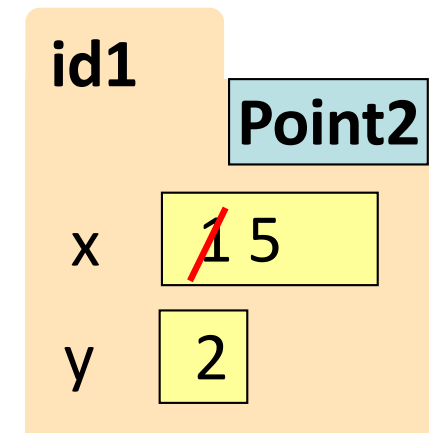
- Boxes for parameters **at the start of the function**
- Boxes for variables local to the function **as they are created**

```
1 def adjust_x(pt, n):  
    pt.x = pt.x + n  
  
    x = 4  
    p = shape.Point2(1,2)  
    adjust_x(p, x)
```

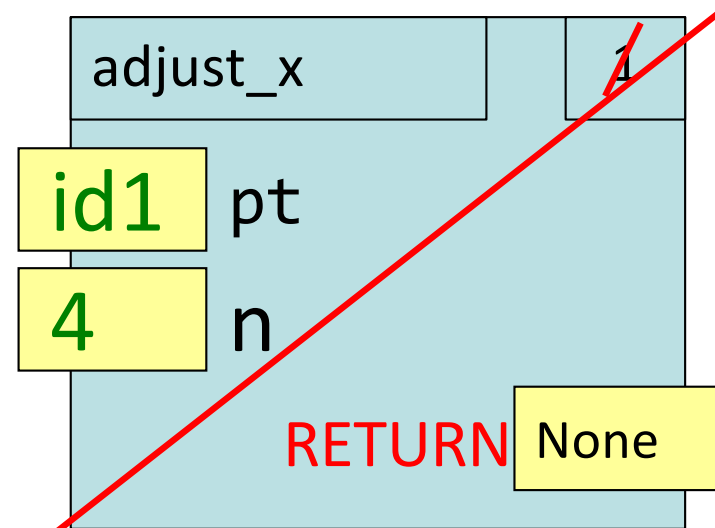
Global Space



Heap Space



Call Stack



Putting it all together

- **Global Space**

- What you “start with”
- Stores global variables
- Lasts until you quit Python

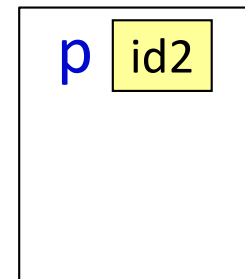
- **Heap Space**

- Where “folders” are stored
- Have to access indirectly

- **Call Frames**

- Parameters
- Other variables local to function
- Lasts until function returns

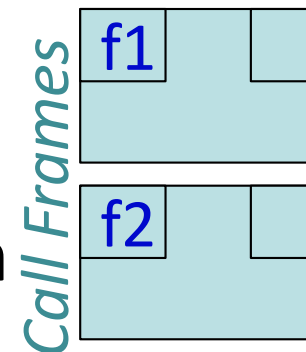
Global Space



Heap Space



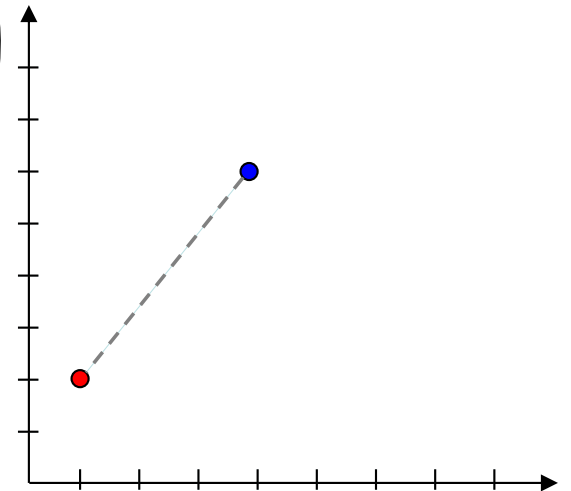
Call Stack



Two Points Make a Line

```
start = shape.Point2(0,0)
stop = shape.Point2(0,0)
print("Where does the line start?")
x = input("x: ")
start.x = int(x)
y = input("y: ")
start.y = int(y)
print("The line starts at (" + x + ", " + y + ").")
print("Where does the line stop?")
x = input("x: ")
stop.x = int(x)
y = input("y: ")
stop.y = int(y)
print("The line stops at (" + x + ", " + y + ").")
```

Where does the line start?
x: 1
y: 2
The line starts at (1,2).
Where does the line stop?
x: 4
y: 6
The line stops at (4,6).



What is `input`? A built-in function.

Start Python interactive mode and type:

```
>>> help(input)           for details!
```

Redundant Code is BAAAAAD!

```
start = shape.Point2(0,0)
stop = shape.Point2(0,0)
print("Where does the line start?")
x = input("x: ")
start.x = int(x)
y = input("y: ")
start.y = int(y)
print("The line starts at (" + x + ", " + y + ").")
print("Where does the line stop?")
x = input("x: ")
stop.x = int(x)
y = input("y: ")
stop.y = int(y)
print("The line stops at (" + x + ", " + y + ").")
```

Let's make a function!

```
# pt is the point object to be initialized
# end type is "start" or "stop"
def configure(pt, end):
    print("Where does the line " + end + "?")
    x = input("x: ")
    pt.x = int(x)
    y = input("y: ")
    pt.y = int(y)
    print("The line " + end + "s at (" + x + ", " + y + ").")
)

start = shape.Point2(0,0)
stop = shape.Point2(0,0)
configure(start, "start")
configure(stop, "stop")
```

Still a bit of redundancy

```
# pt is the point object to be initialized
# end type is "start" or "stop"
def configure(pt, end):
    print("Where does the line " + end + "?")
    x = input("x: ")
    pt.x = int(x)
    y = input("y: ")
    pt.y = int(y)
    print("The line " + end + "s at (" + x + ", " + y + ")."
)

start = shape.Point2(0,0)
stop = shape.Point2(0,0)
configure(start, "start")
configure(stop, "stop")
```

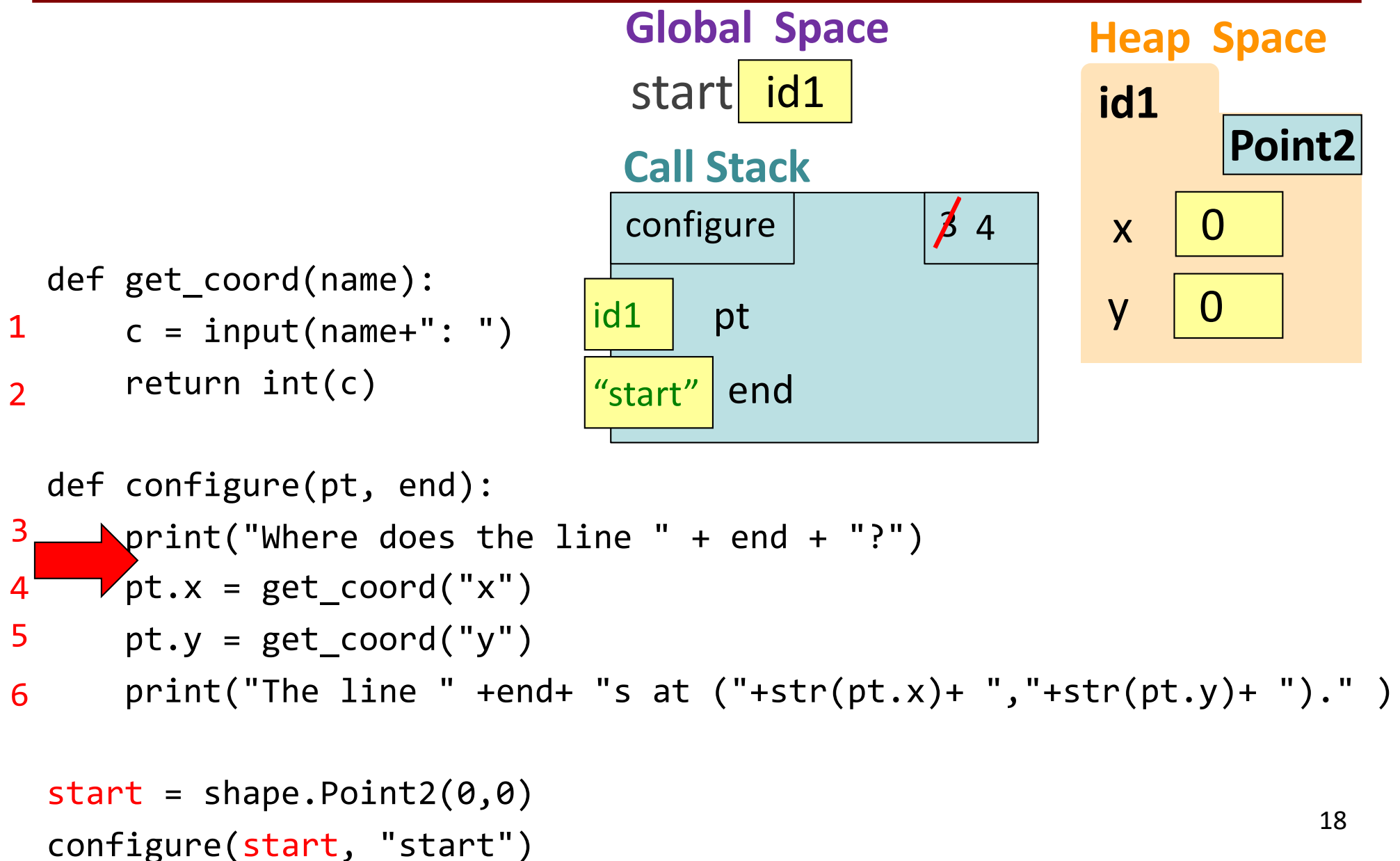
Yay, Helper Functions!

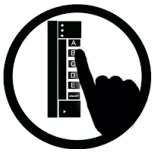
```
def get_coord(name):  
    x = input(name+": ")  
    return int(x)  
  
def configure(pt, end):  
    print("Where does the line " + end + "?")  
    pt.x = get_coord("x")  
    pt.y = get_coord("y")  
    print("The line " +end+ "s at (" +str(pt.x)+ ", "+str(pt.y)+ ")." )  
)  
  
start = shape.Point2(0,0)  
stop = shape.Point2(0,0)  
configure(start, "start")  
configure(stop, "stop")
```

Frames and Helper Functions

- Functions can call each other!
- Each call creates a *new call frame*
- Writing the same several lines of code in 2 places? Or code that accomplishes some conceptual sub-task? Or your function is getting too long? Write a **helper function!** Makes your code easier to
 - **read**
 - **write**
 - **edit**
 - **debug**

Drawing Frames for Helper Functions (1)





Q1: what do you do next?

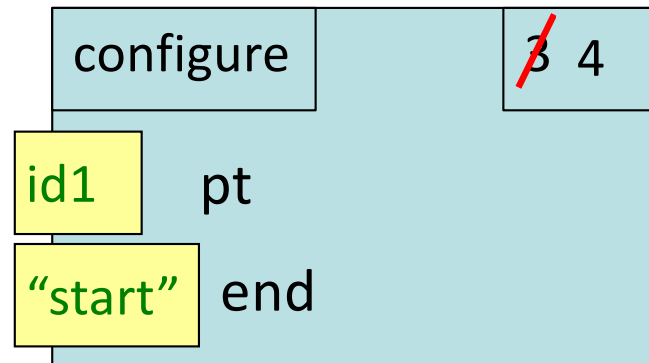
```
def get_coord(name):  
1   c = input(name+": ")  
2   return int(c)  
  
def configure(pt, end):  
3   print("Where does the line start?")  
4   pt.x = get_coord("x")  
5   pt.y = get_coord("y")  
6   print("The line " + end + " is configured.")
```

```
start = shape.Point2(0,0)  
configure(start, "start")
```

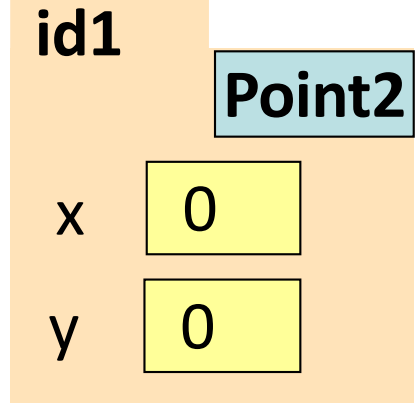
Global Space

start id1

Call Stack



Heap Space



- A: Cross out the configure call frame.
- B: Create a get_coord call frame.
- C: Cross out the 4 in the call frame.
- D: A & B
- E: B & C

Drawing Frames for Helper Functions (2)

A
B CORRECT
C
D
E

```
1 def get_coord(name):  
2     c = input(name+": ")  
3     return int(c)
```

```
3 def configure(pt, end):  
4     print("Where does the line start?")  
5     pt.x = get_coord("x")  
6     pt.y = get_coord("y")  
7     print("The line " +end+ " " +str(pt.x)+ " " +str(pt.y)+ " ")."
```

```
start = shape.Point2(0,0)  
configure(start, "start")
```

Global Space

start id1

Call Stack

configure

~~3~~ 4

id1

pt

"start"

end

get_coord

1

"x"

name

Heap Space

id1

Point2

x

0

y

0

Not done!
Do not
cross out!!

Drawing Frames for Helper Functions (3)

Assume user types 1
at Python shell
prompt

```
def get_coord(name):
1  c = input(name+": ")
2  return int(c)
```

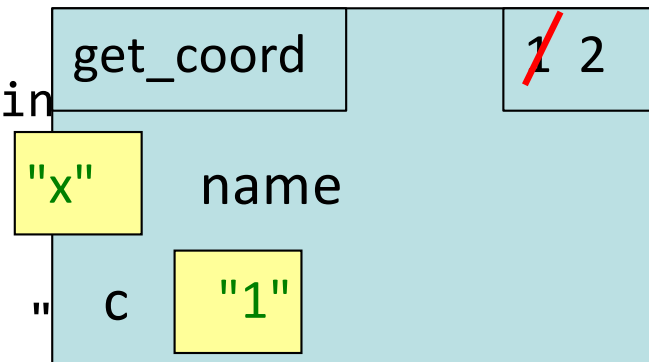
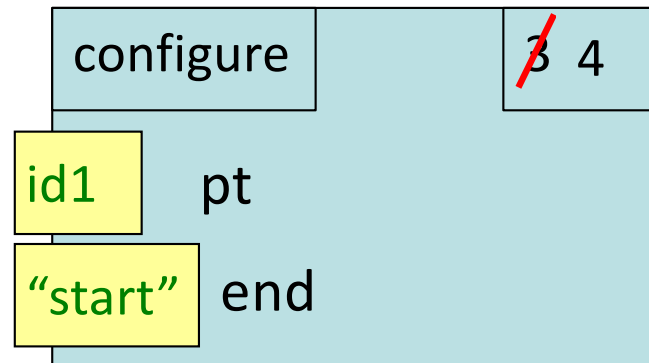
```
def configure(pt, end):
3  print("Where does the line start? ")
4  pt.x = get_coord("x")
5  pt.y = get_coord("y")
6  print("The line " +end+ " " +str(pt.x)+ " " +str(pt.y)+ " ")."
```

```
start = shape.Point2(0,0)
configure(start, "start")
```

Global Space

start id1

Call Stack



Heap Space

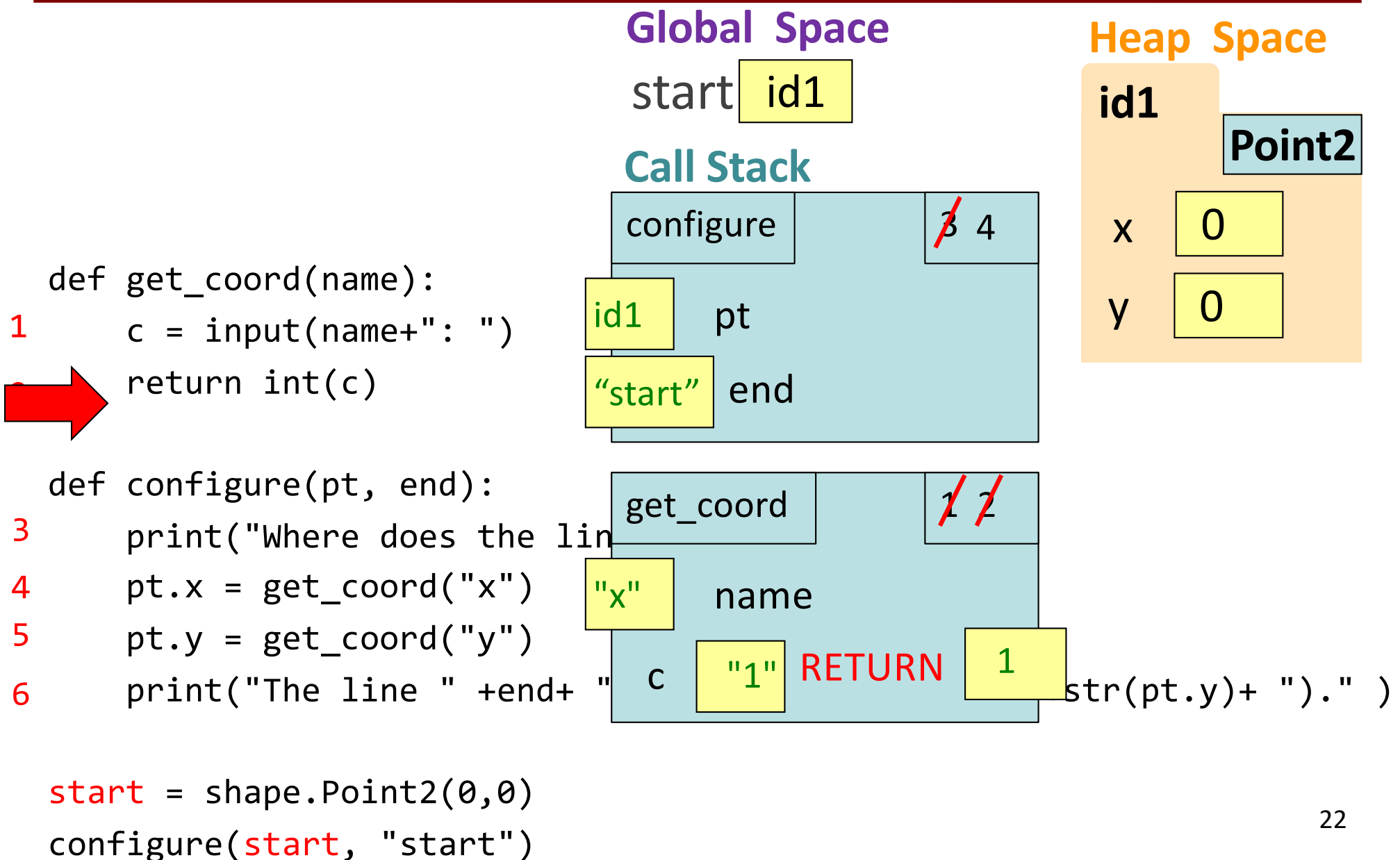
id1

Point2

x 0

y 0

Drawing Frames for Helper Functions (4)



Drawing Frames for Helper Functions (5)



To do: Finish the diagram, assuming that user types 2 at Python shell prompt when this `get_coord` call executes.

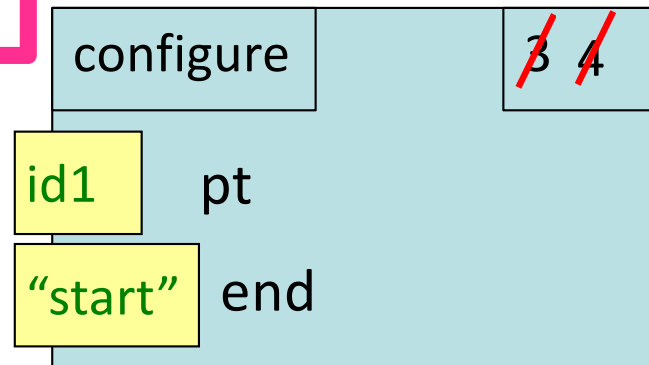
```
def get_coord(name):
1   c = input(name+": ")
2   return int(c)

def configure(pt, end):
3   print("Where does the line start?")
4   pt.x = get_coord("x")
5   pt.y = get_coord("y")
6   print("The line " + end + " is at (" + str(pt.x) + ", " + str(pt.y) + ")." )
```

Global Space

start id1

Call Stack



Heap Space

id1

Point2

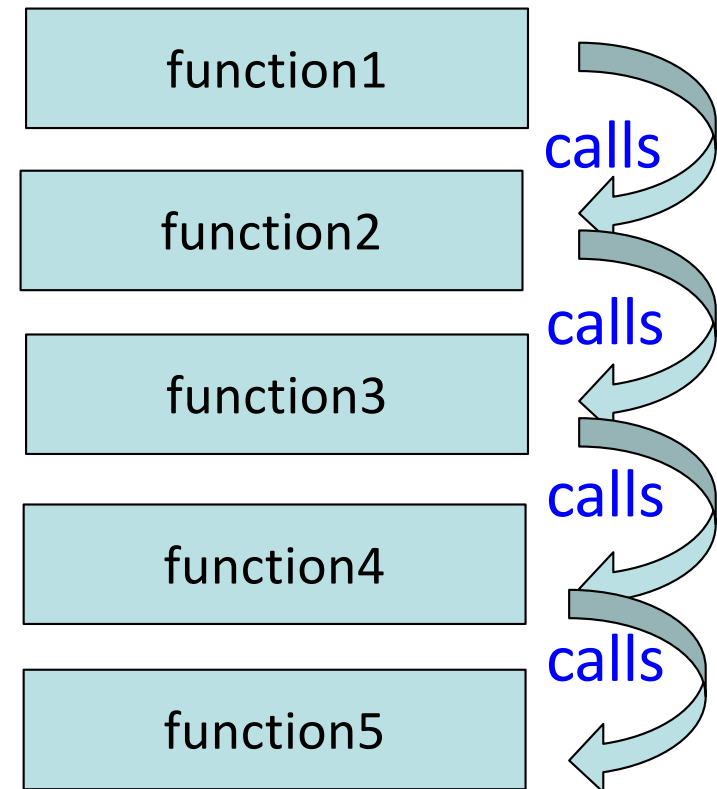
x ~~0~~ 1

y 0

```
start = shape.Point2(0,0)
configure(start, "start")
```


The Call Stack

- The set of function frames drawn in call order
- Functions frames are “stacked”
 - Cannot remove one above w/o removing one below
- Python must keep the **entire stack** in memory
 - Error if it cannot hold stack (“stack overflow”)



Errors and the Call Stack

```
def get_coord(name):  
9   c = input(name+": ")  
10  return int(x)  
  
def configure(pt, end):  
13  print("Where does the line "  
14  pt.x = get_coord("x")  
15  pt.y = get_coord("y")  
16  print("The line " +end+ "s at (" +x+ ", "+y+ ")." )  
  
18 start = shape.Point2(0,0)  
19 configure(start, "start")
```

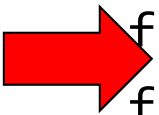


```
Where does the line start?  
x: 1  
Traceback (most recent call last):  
  File "v3.py", line 19, in <module>  
    configure(start, "start")  
  File "v3.py", line 14, in configure  
    pt.x = get_coord("x")  
  File "v3.py", line 10, in get_coord  
    return str(x)  
NameError: name 'x' is not defined
```

Q2: what does the call stack look like at this point in the execution of the code?

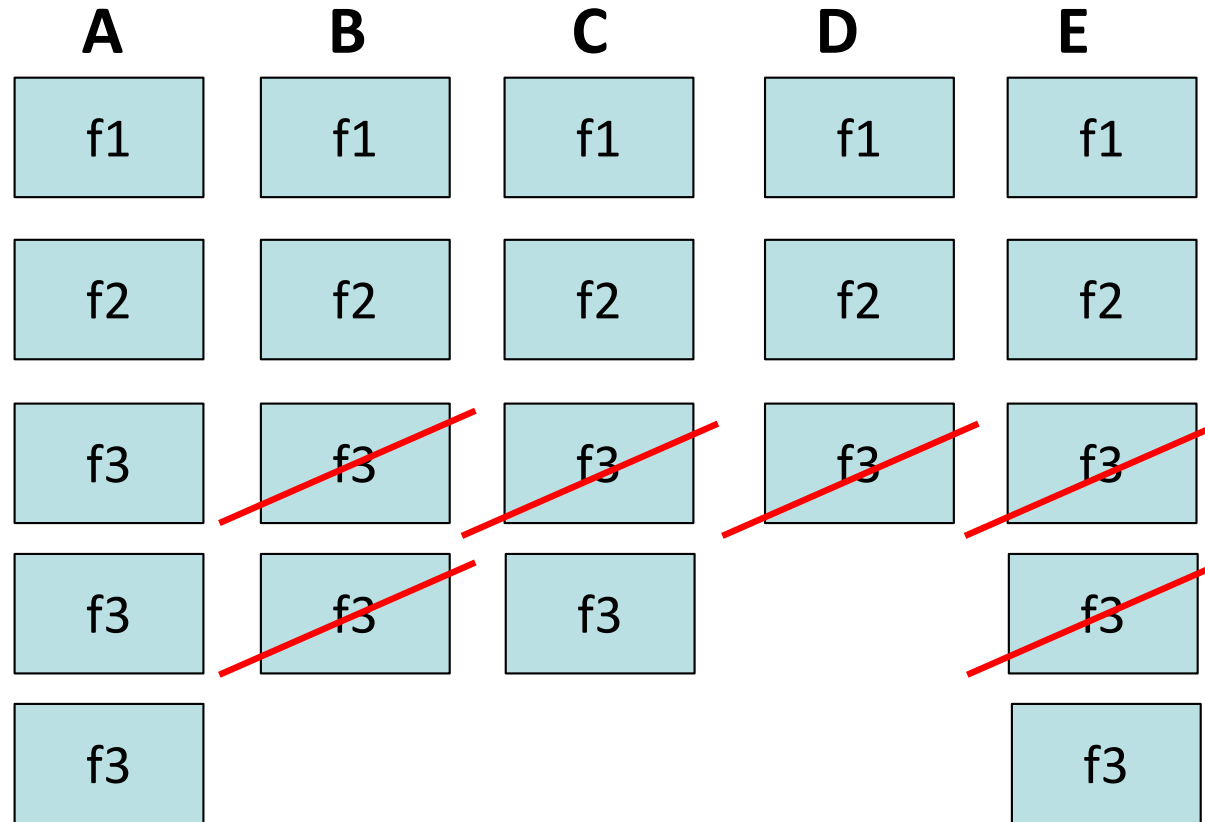
```
def f3():  
    print("f3")
```

```
def f2():  
    print("f2")  
    f3()
```

 `f3()`
`f3()`

```
def f1():  
    print("f1")  
    f2()
```

`f1()`



Modules and Global Space

Import

- Creates a global **variable** (same name as module)
- Puts variables, functions of module in a **folder**
- Puts folder id in the global **variable**

Global Space

math

id5

Heap Space

id5

math module

pi

3.141592

e

2.718281

functions

```
>>> import math
```


Modules vs Objects

```
>>> import math  
>>> math.pi
```

```
>>> p = shapes.Point3(5,2,3)  
>>> p.x
```

Global Space

math id5

p id3

shapes id5

*assumes we
imported shapes
(not shown)*

Heap Space

id1

math module

pi 3.141592

e 2.718281

functions

id3

Point3

x 5

y 2

z 3

Functions and Global Space

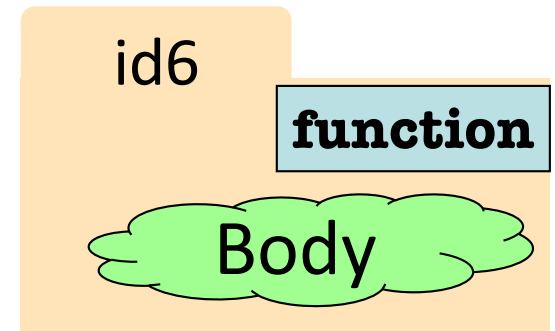
A function definition

- Creates a global variable (same name as function)
- Creates a **folder** for body
- Puts folder id in the global variable

Global Space

INCH_PER_FT **12**
get_feet **id6**

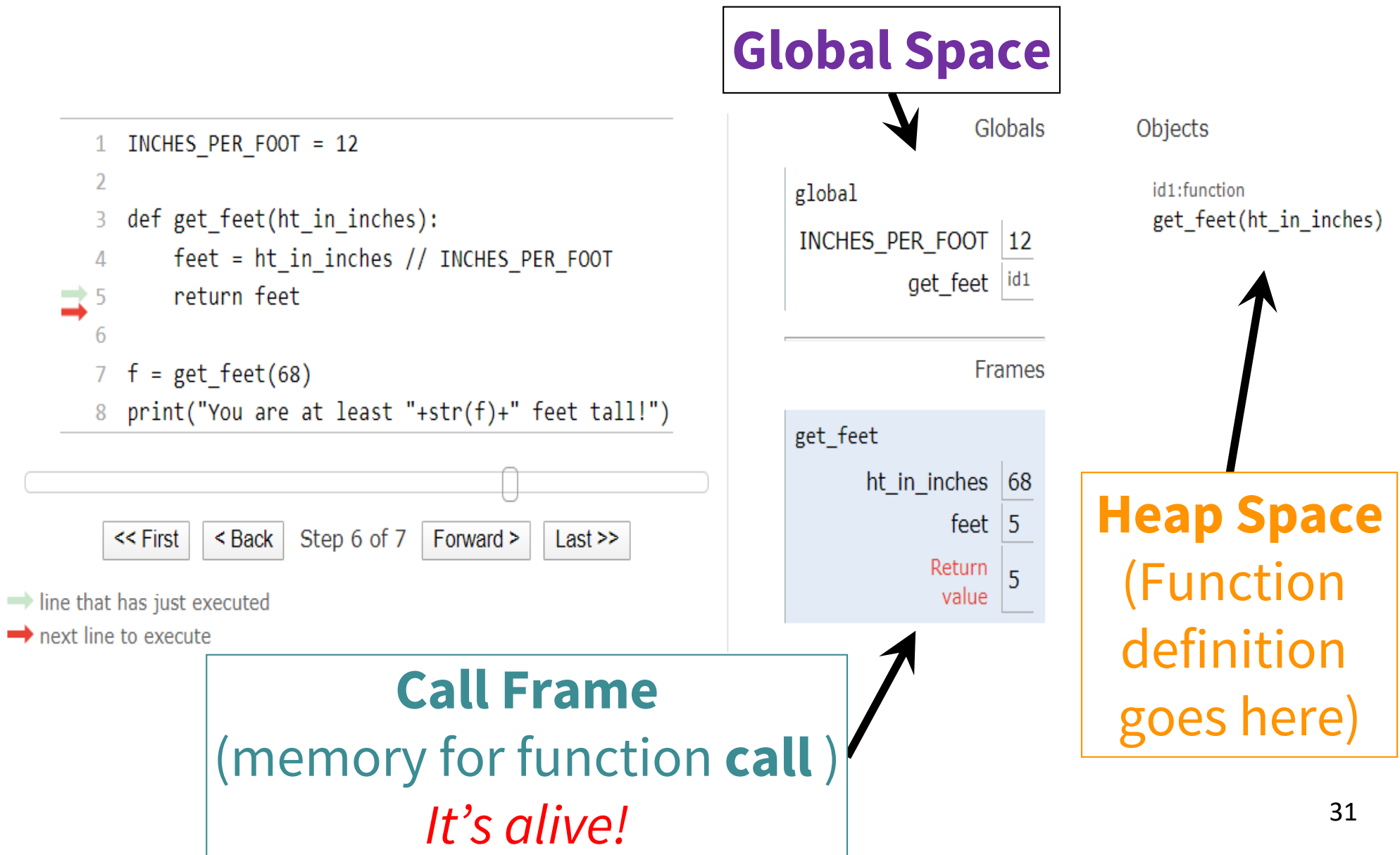
Heap Space



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

Body

Function Definition vs. Call Frame



Storage in Python

- **Global Space**

- What you “start with”
- Stores global variables, modules & functions
- Lasts until you quit Python

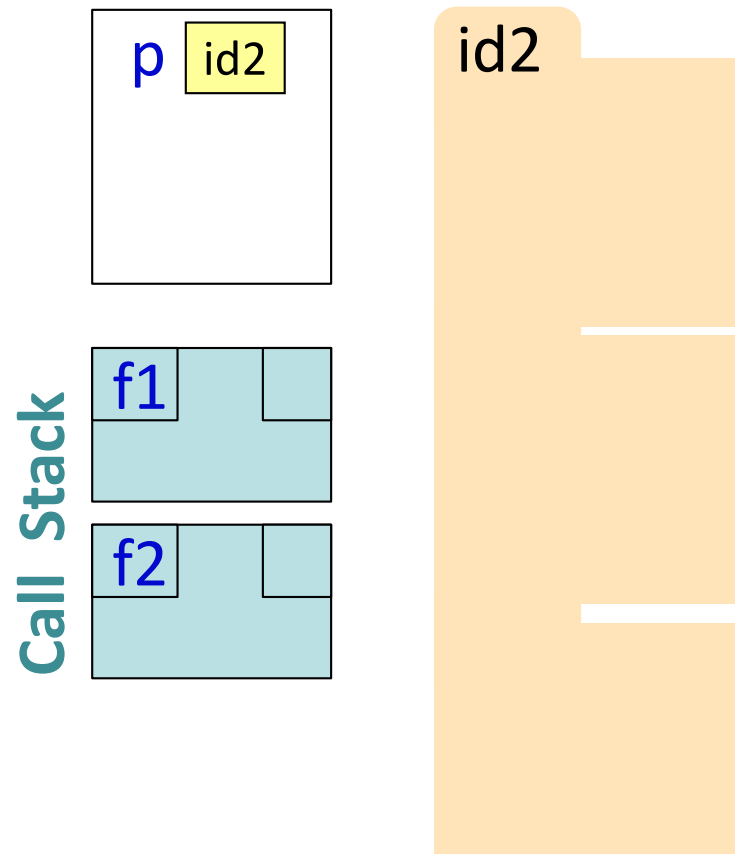
- **Heap Space**

- Where “folders” are stored
- Have to access indirectly

- **Call Stack**

- Where Call Frames live
- Parameters
- Other variables local to function
- Lasts until function returns

Global Space **Heap Space**



Don't draw module folder, function folder

Folders that we **do not require you to draw**:

- Module folder is created upon `import`, for example,
`import math`
- Function folder is created with `def` (the function header), for example,
`def get_feet(height_in_inches):`

Don't draw those folders and the variables that store their ids; we only explained those folders to explain what you see in Python Tutor.

Do not draw them.

Q3: what does the call stack look like at this point in the execution of the code?

```
def f3():  
    print("f3")
```

```
def f2():  
    print("f2")  
    f3()  
    f3()  
    f3()
```

```
def f1():  
    print("f1")  
    f2()
```

```
f1()
```

