http://www.cs.cornell.edu/courses/cs1110/2022sp

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]

Announcements

- Last day to inform us of your Prelim 1 conflict!
- Previous Exams located on the website

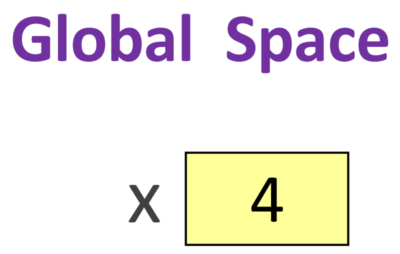
- A1 revision process: A1 closed now on CMS for grading. Set your CMS notifications to "receive email when …" When feedback is released, expected on late Thursday, Feb 24 afternoon, read *re*submission instructions

- A2 to be released today

Global Space

Global Space

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



```
x = 4
```

Enter Heap Space

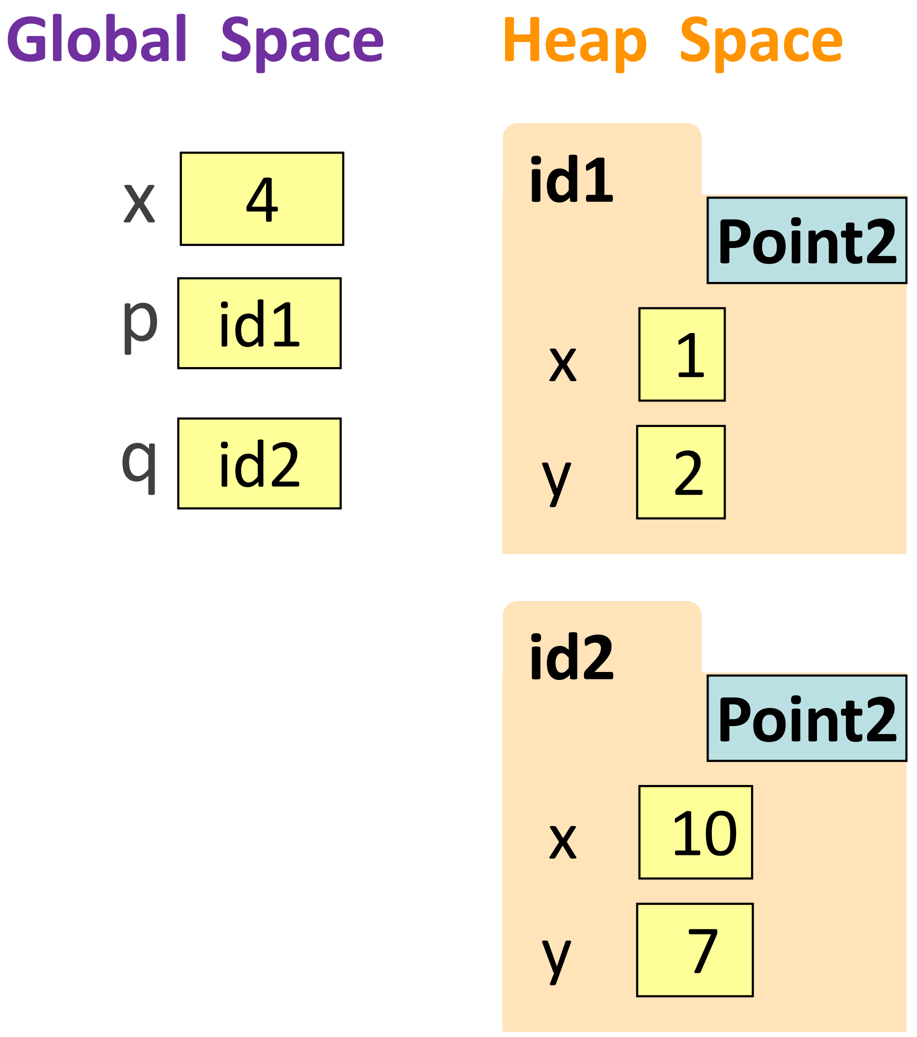
Global Space

- 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)
```



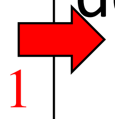
p & **q** live in Global Space. Their folders live on the Heap.

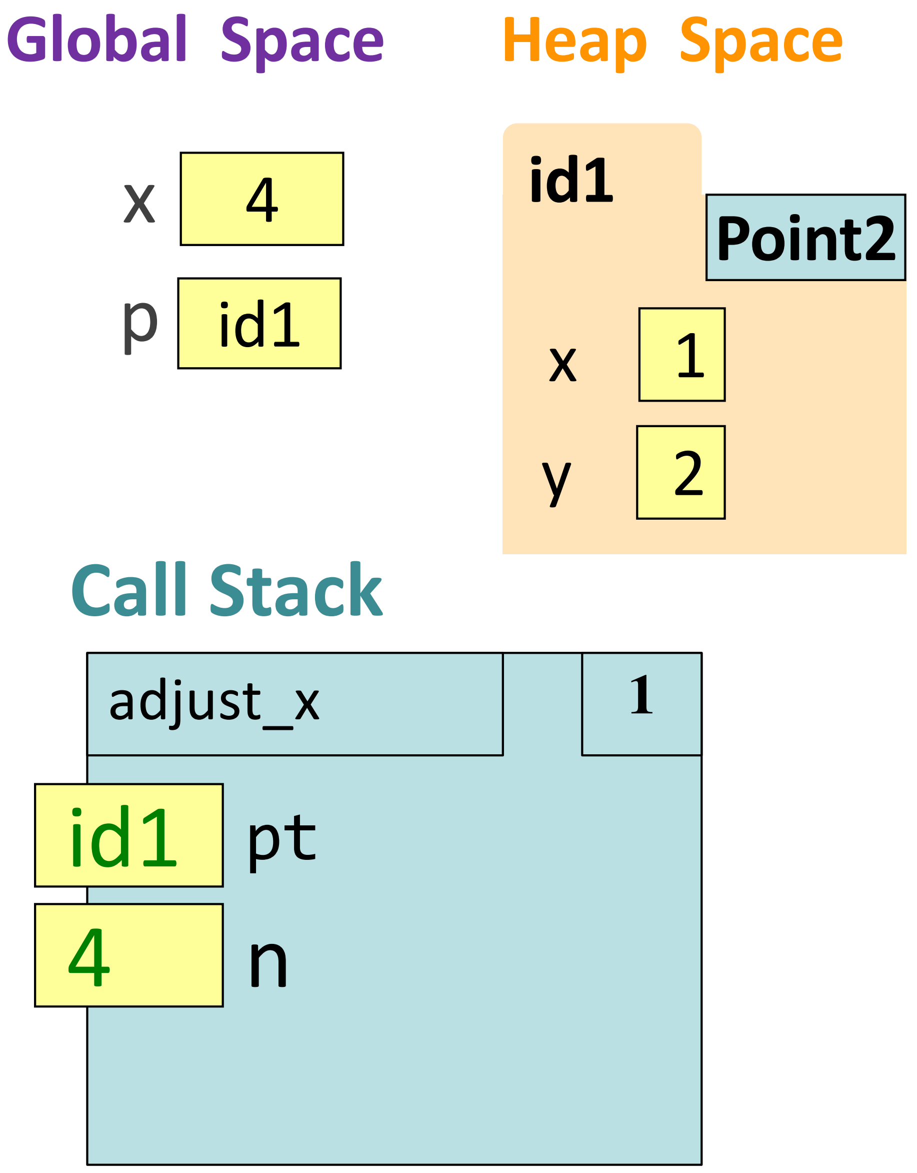
Calling a Function Creates a Call Frame (1)

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**

```
def adjust_x(pt, n):
    pt.x = pt.x + n

x = 4
p = shape.Point2(1,2)
adjust_x(p, x)
```



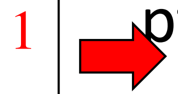


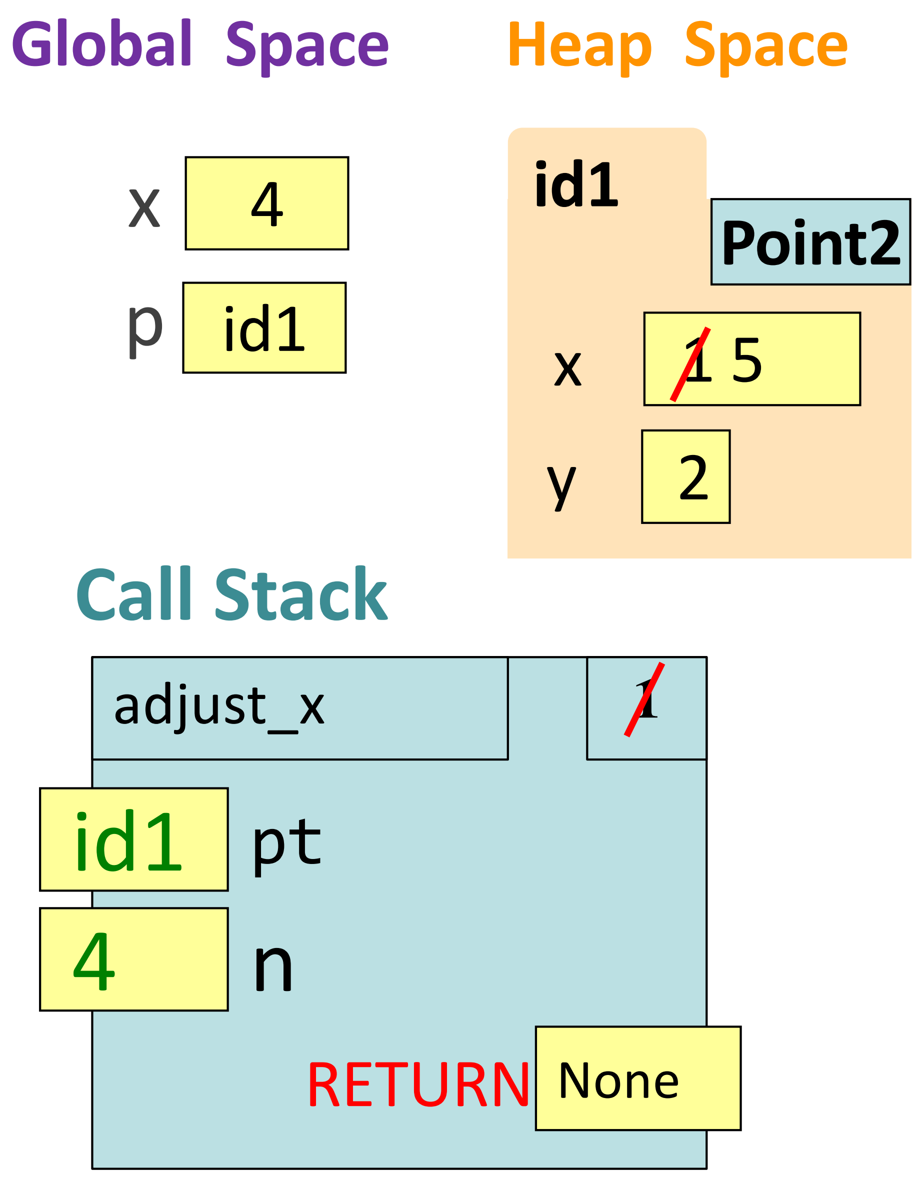
Calling a Function Creates a Call Frame (2)

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):
    pt.x = pt.x + n

x = 4
p = shape.Point2(1,2)
adjust_x(p, x)
```



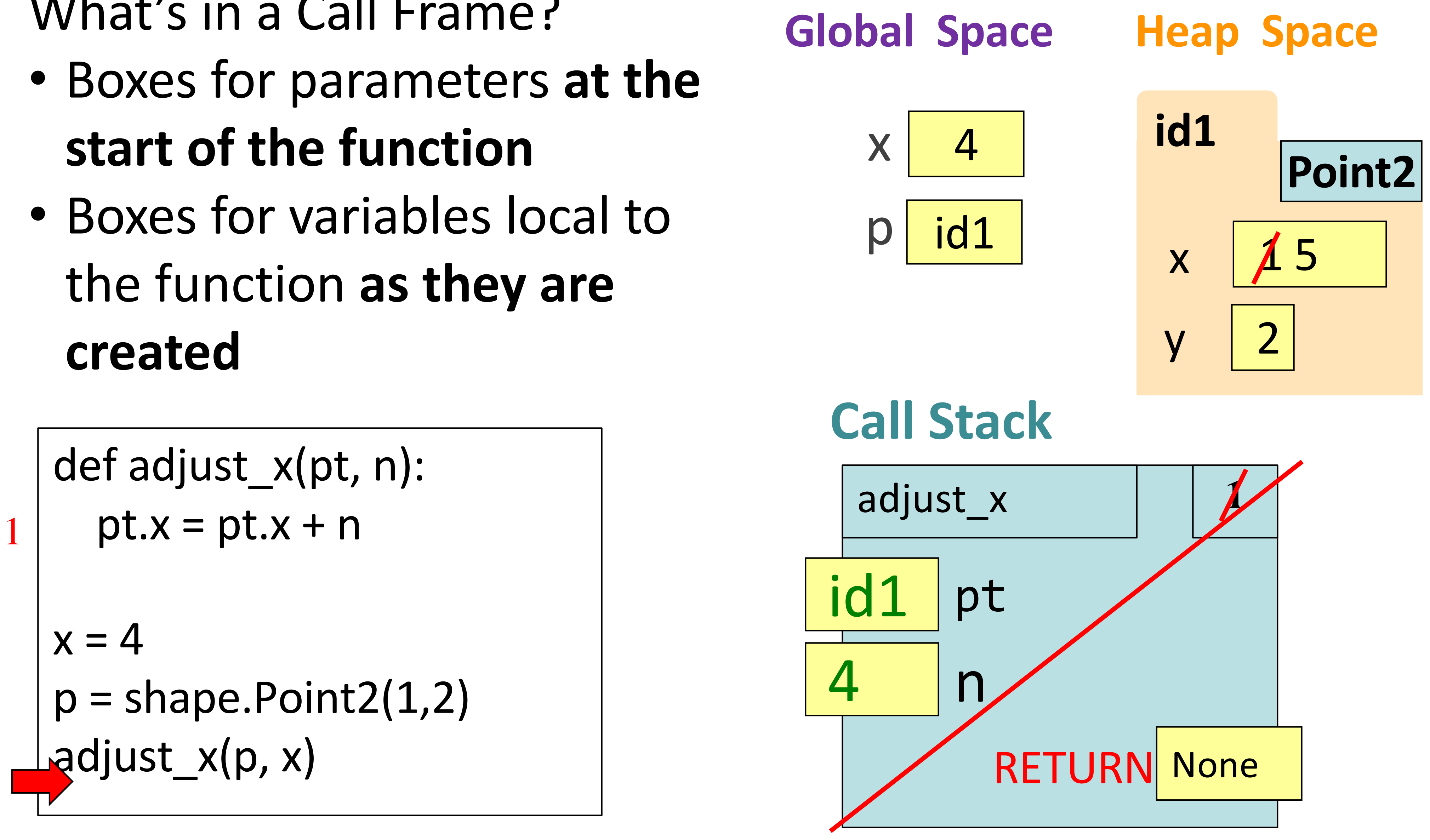


Calling a Function Creates a Call Frame (3)

What's in a Call Frame?

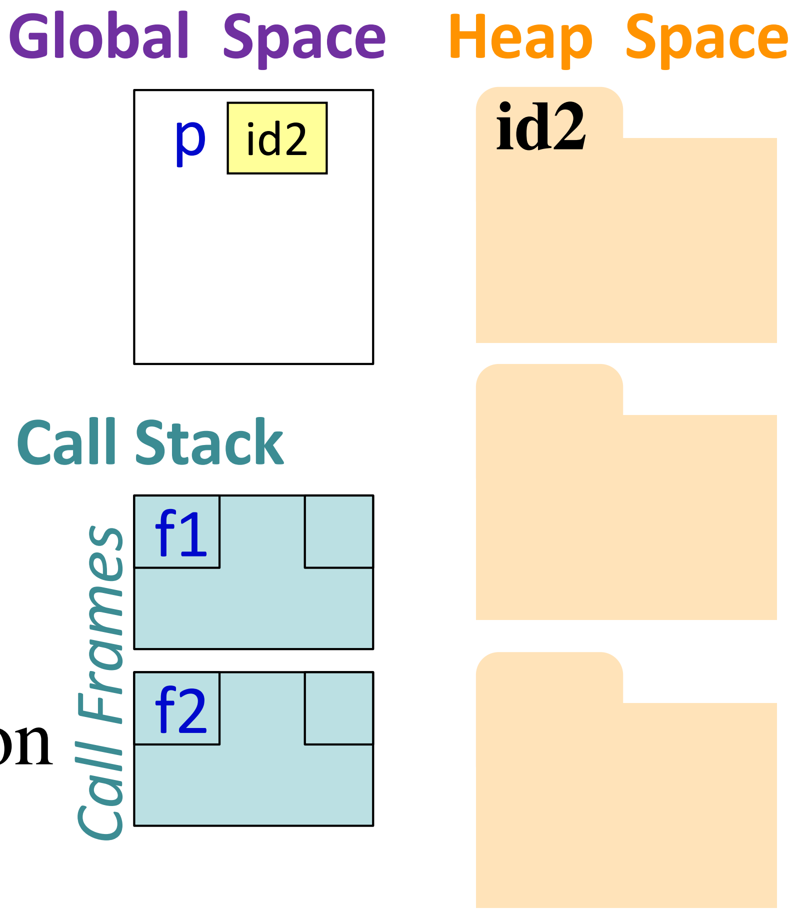
- Boxes for parameters **at the start of the function**
- Boxes for variables local to the function **as they are created**

```
def adjust_x(pt, n):
    pt.x = pt.x + n

x = 4
p = shape.Point2(1,2)
adjust_x(p, x)
```



Putting it all together

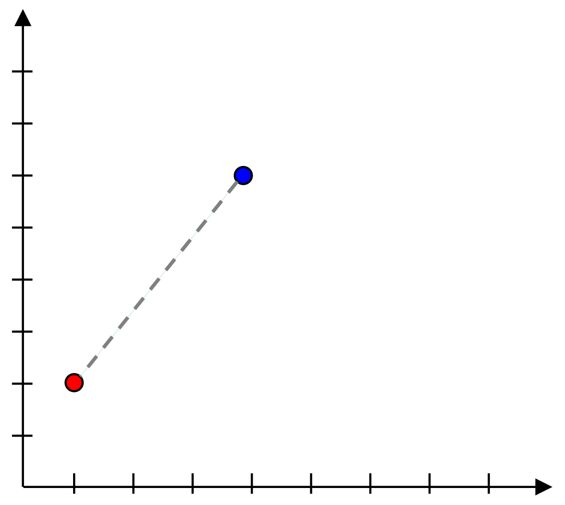
- **Global Space**
  - What you "start with"
  - Stores global variables
  - Lasts until you quit Python
- **Heap Space**
  - Where "folders" are stored
  - Have to access indirectly
- **Call Frames**
  - Parameters
  - Other variables local to function
  - Lasts until function returns



Two Points Make a Line

```python
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).
```



Redundant Code is BAAAD!

```
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")
```

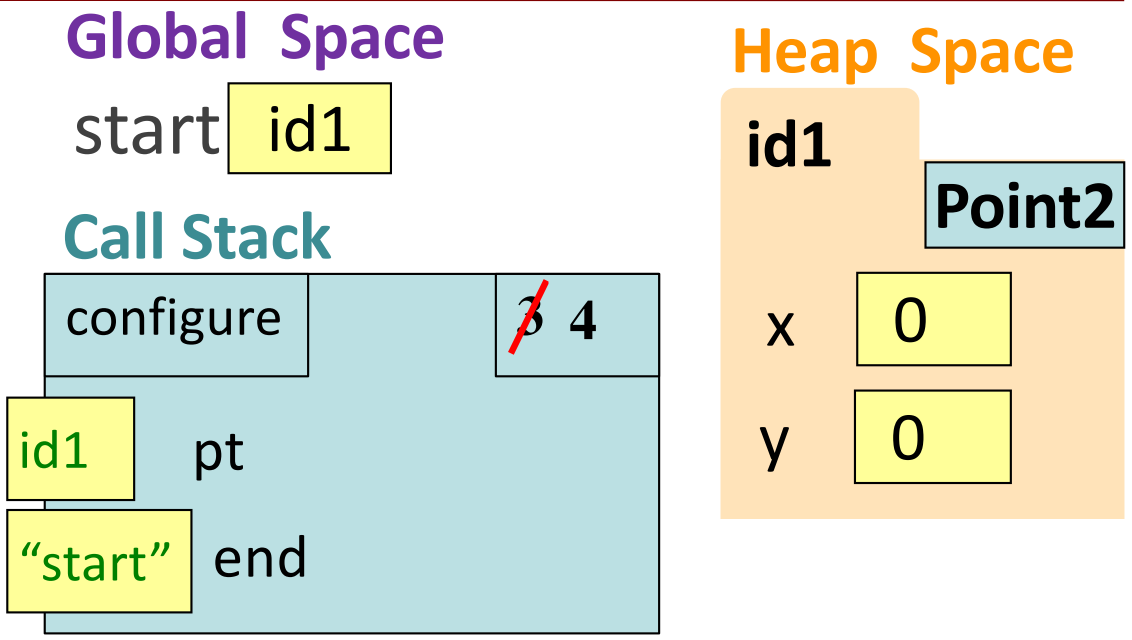
Yay, Helper Functions!

```
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**

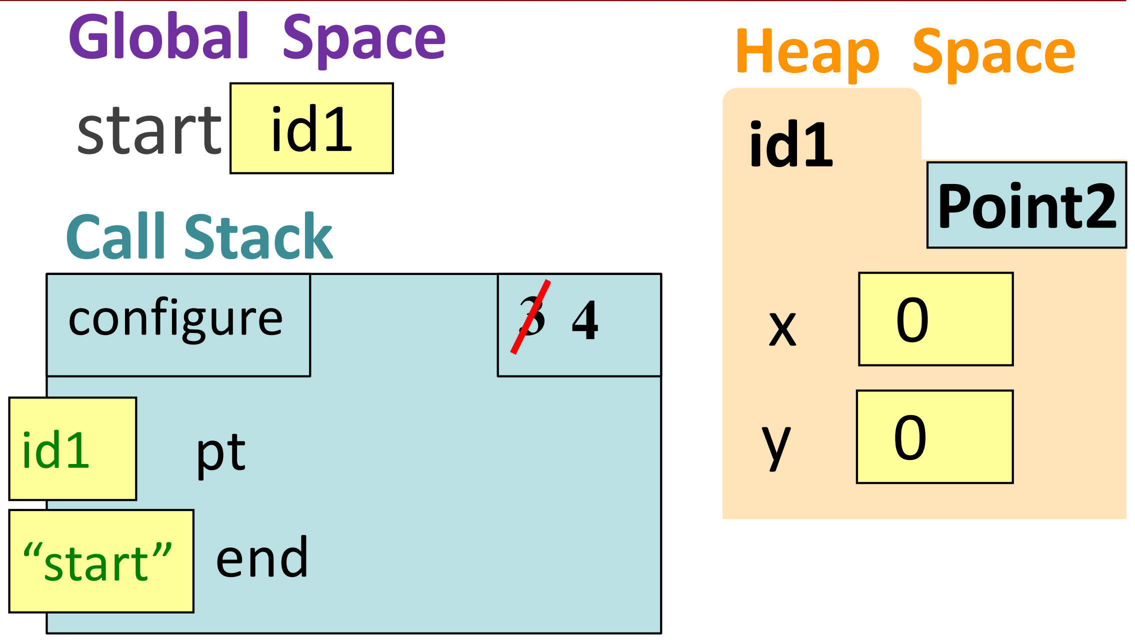
Drawing Frames for Helper Functions (1)



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

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

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

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 lin
4     pt.x = get_coord("x")
5     pt.y = get_coord("y")
6     print("The line " +end+ "

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

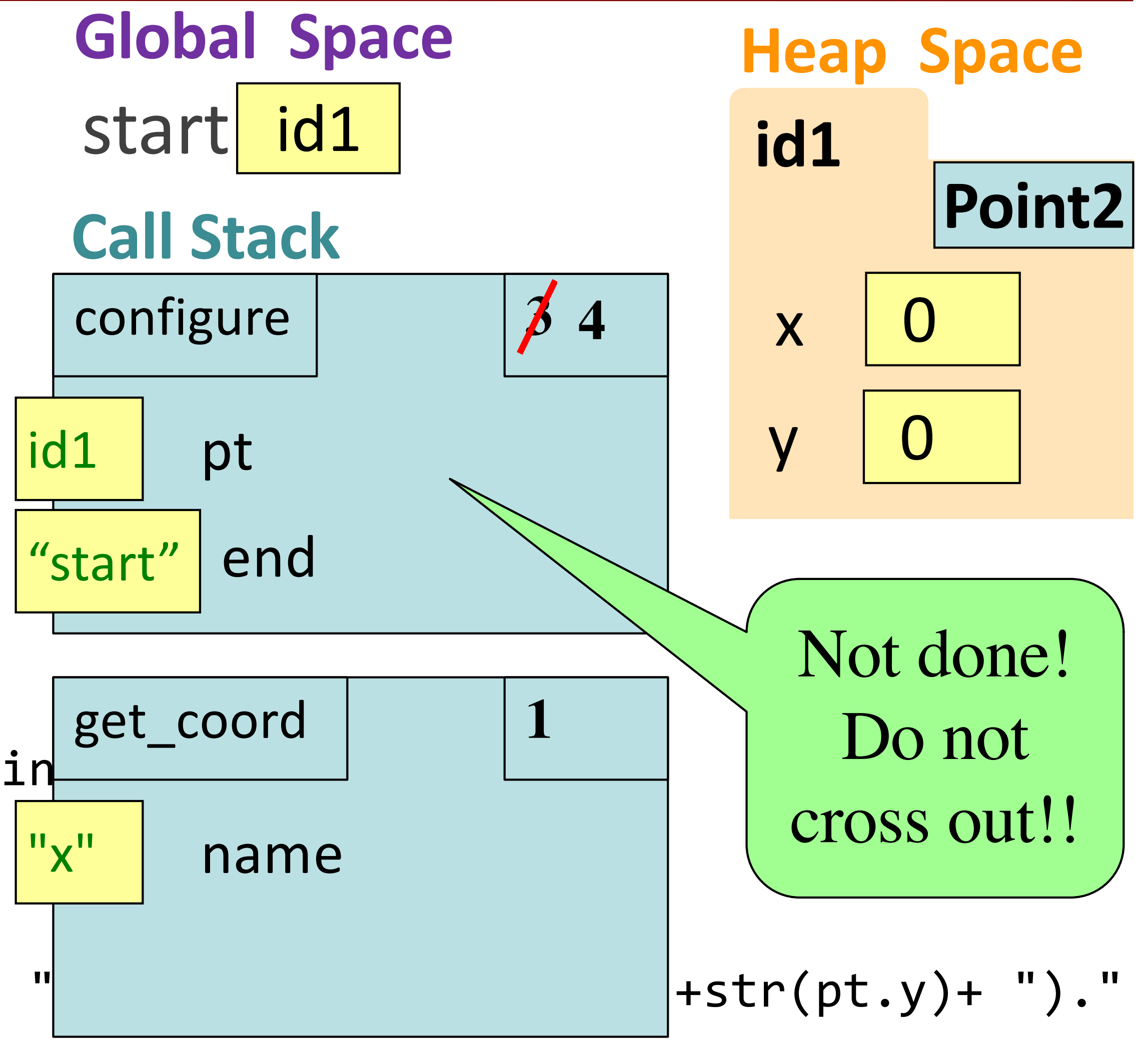


A: Cross out the configure call frame.
B: Create a get_coord call frame.
C: Cross out the 4 in the call frame.
D: A & B
E: B & C

Drawing Frames for Helper Functions (2)

A
B **CORRECT**
C
D
E

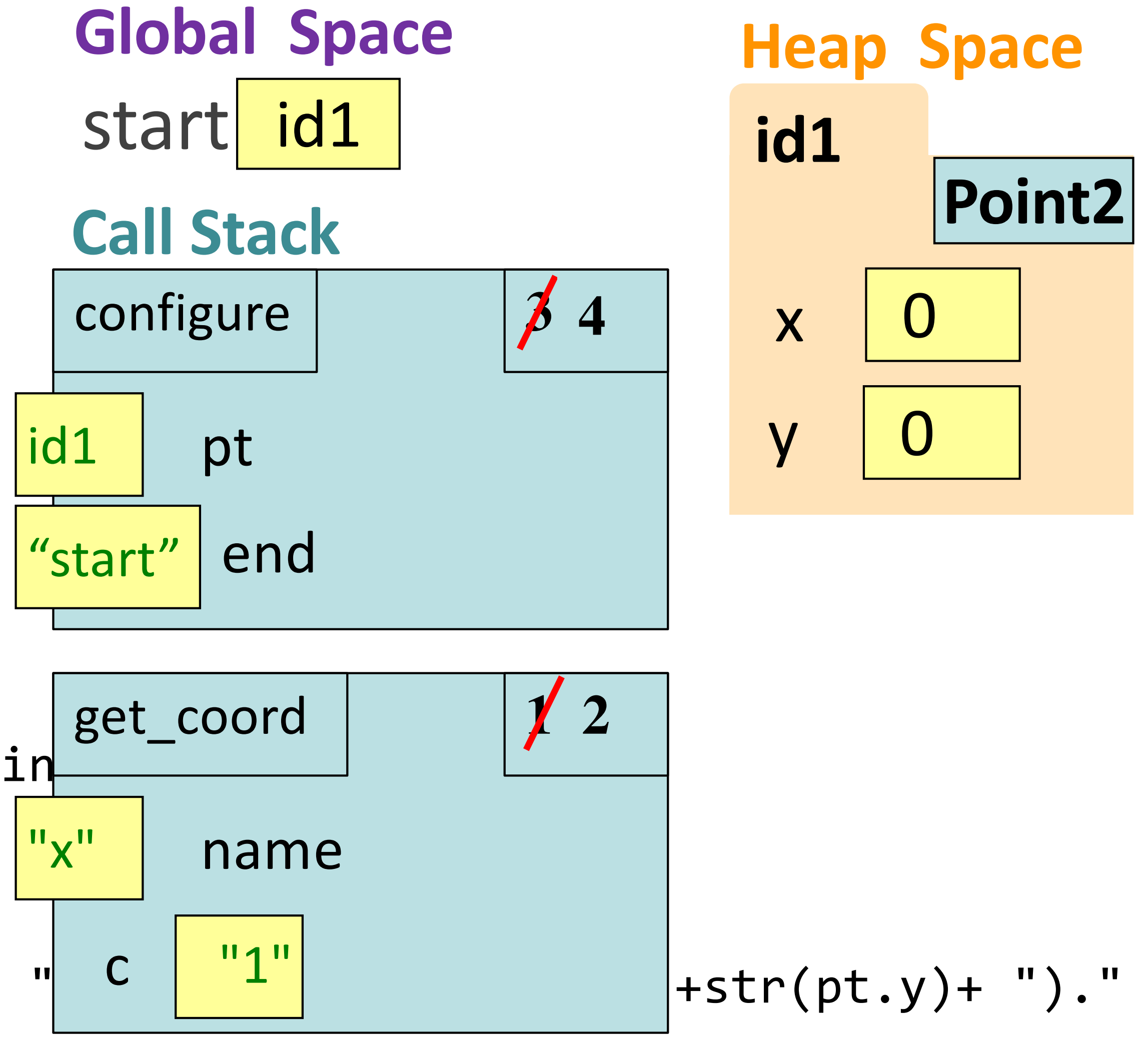
```
  def get_coord(name):
1     c = input(name+": ")
2     return int(c)

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

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



Drawing Frames for Helper Functions (3)

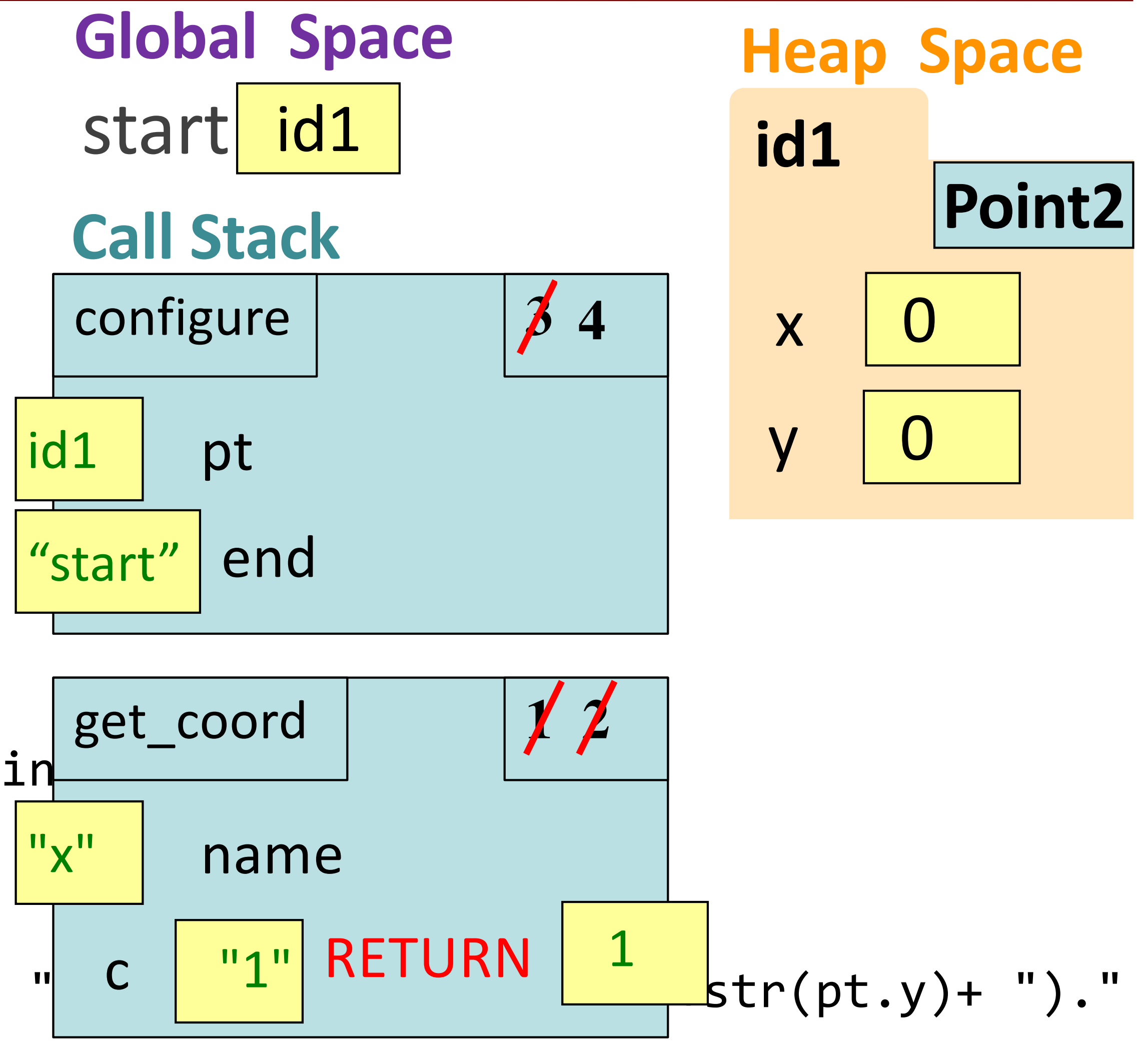
Assume user types **1** *at Python shell prompt*

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

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

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



Drawing Frames for Helper Functions (4)

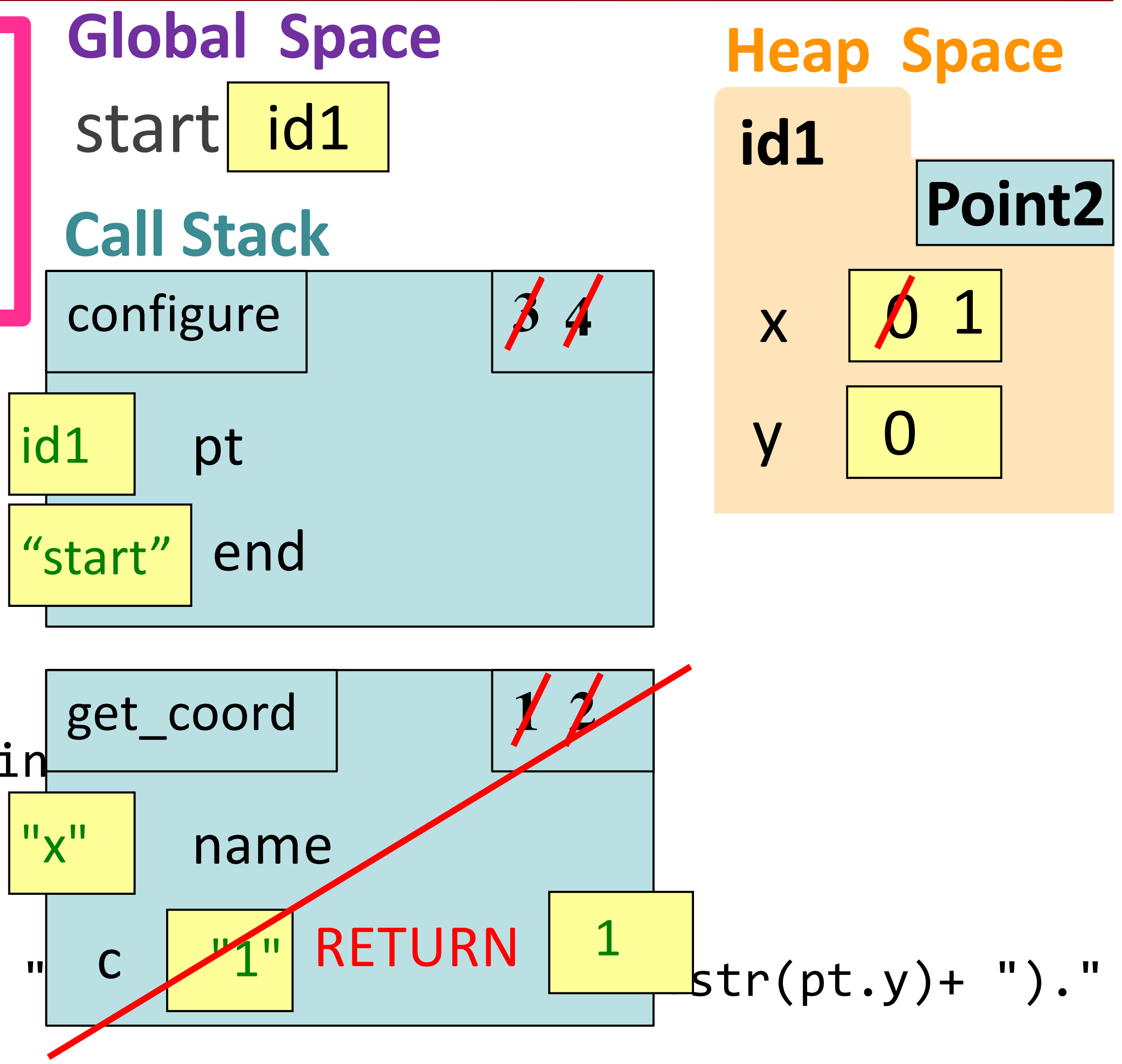
```
  def get_coord(name):
1     c = input(name+": ")
2     return int(c)

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

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



Drawing Frames for Helper Functions (5)

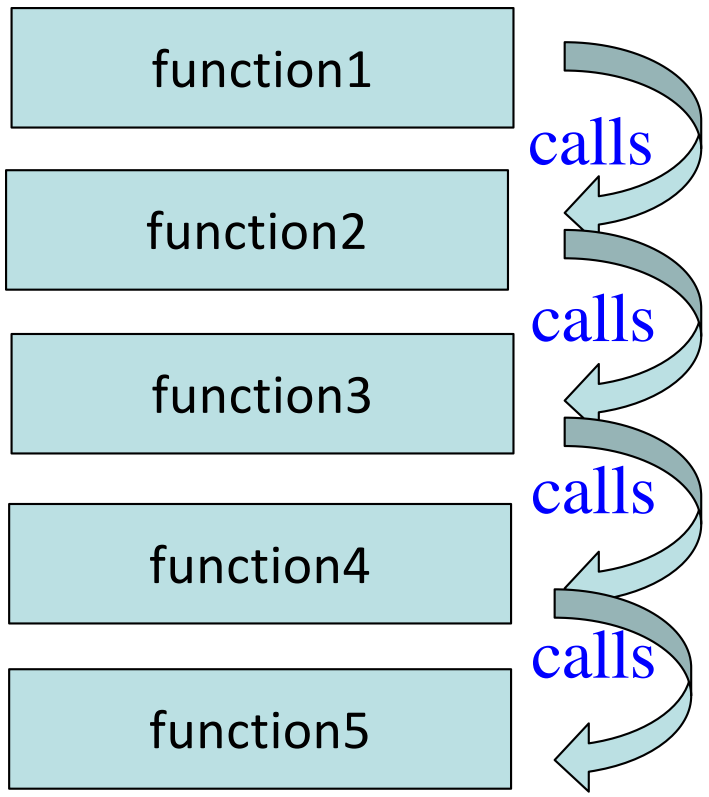
To do: Finish the diagram, assuming that user types **2** *at Python shell prompt when this* get_coord *call executes.*

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

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

  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")



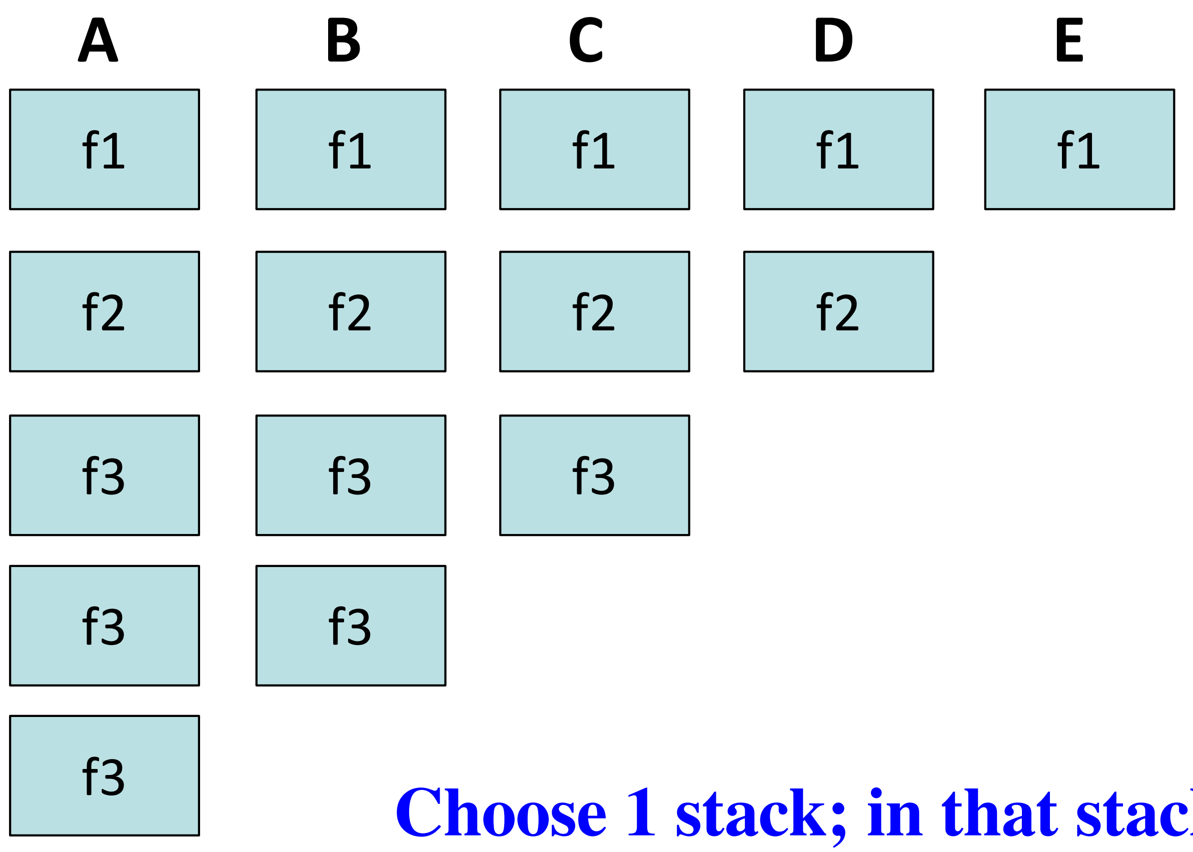
Errors and the Call Stack

```
   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?

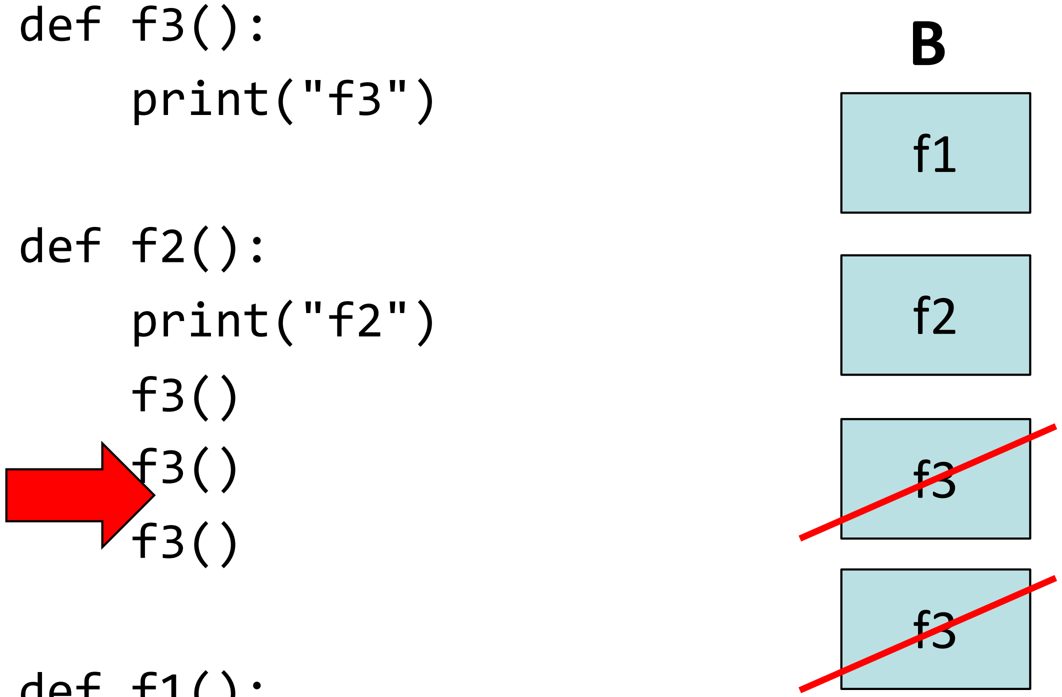
```
def f3():
    print("f3")

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

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

f1()
```



Choose 1 stack; in that stack cross out any frames that should have ended.

A2: 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()
```

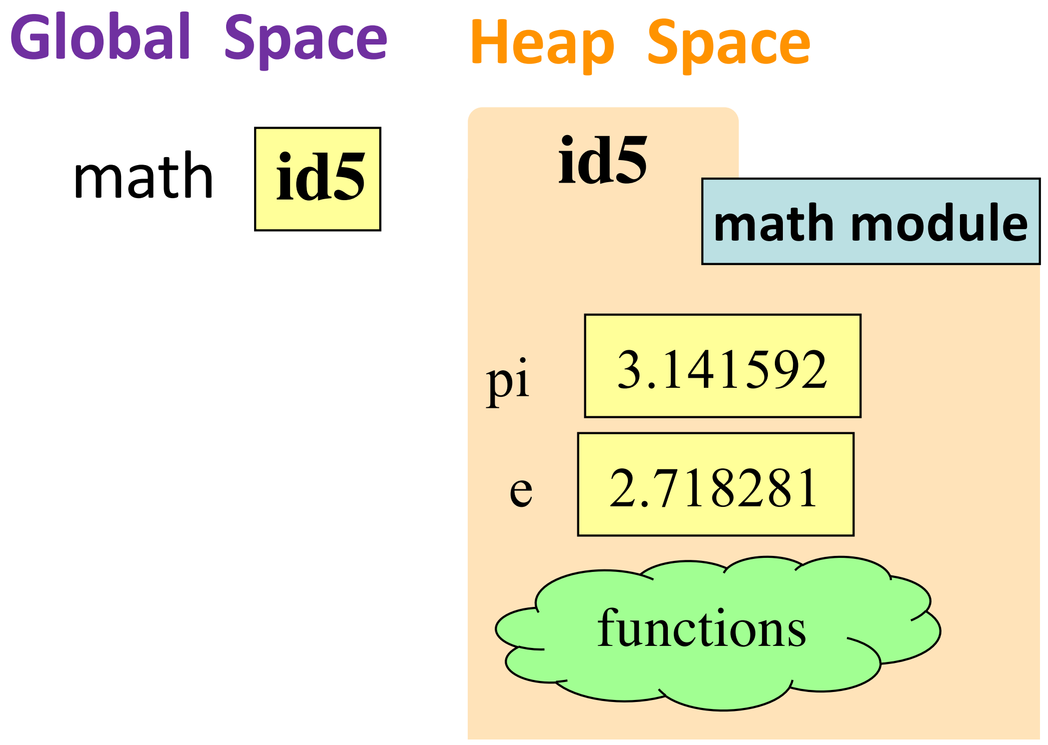


Choose 1 stack; in that stack cross out any frames that should have ended.

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

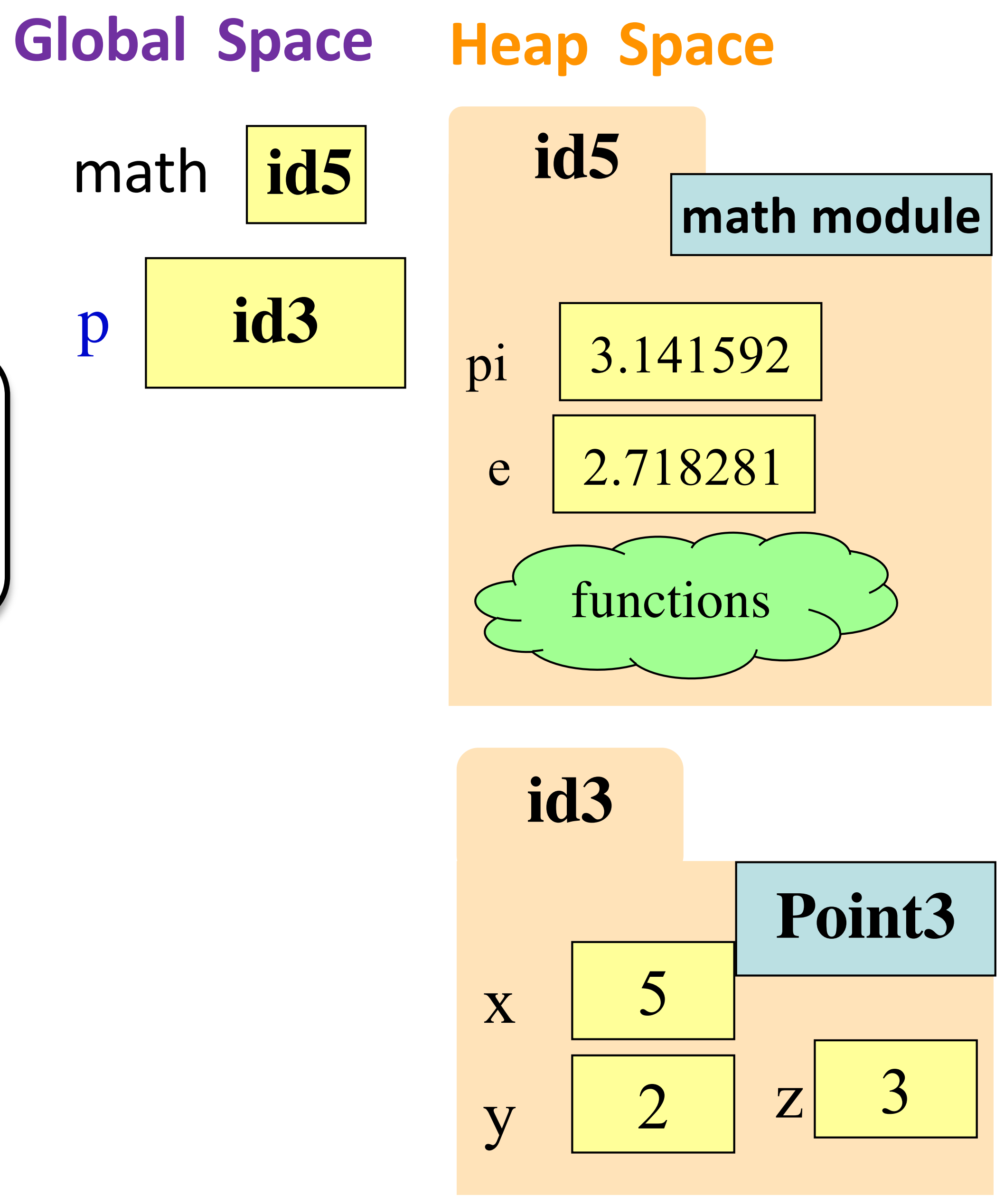


```
>>> import math
```

Modules vs Objects

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

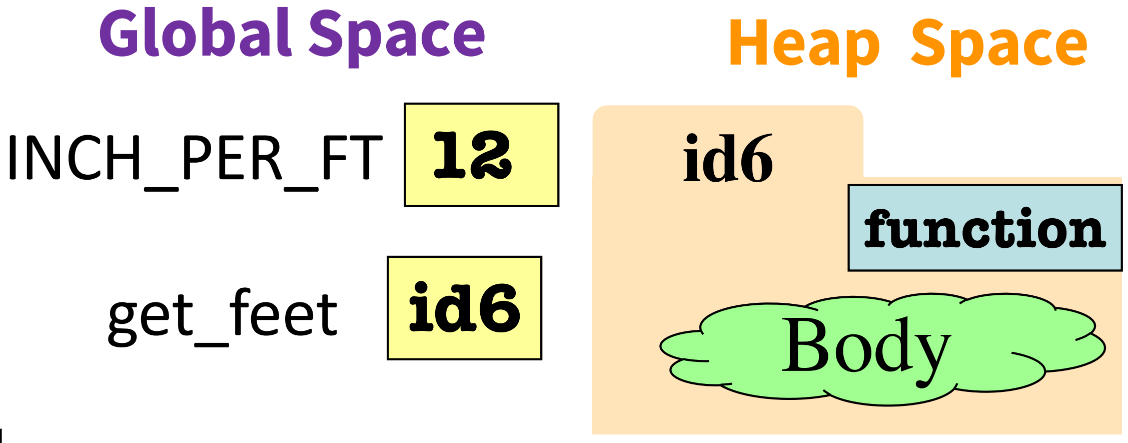
```
>>> p = shapes.Point3(5,2,3)
>>> p.x
```



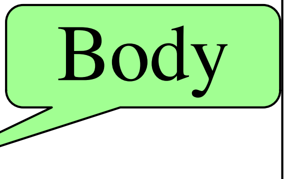
Functions and Global Space

A function definition

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

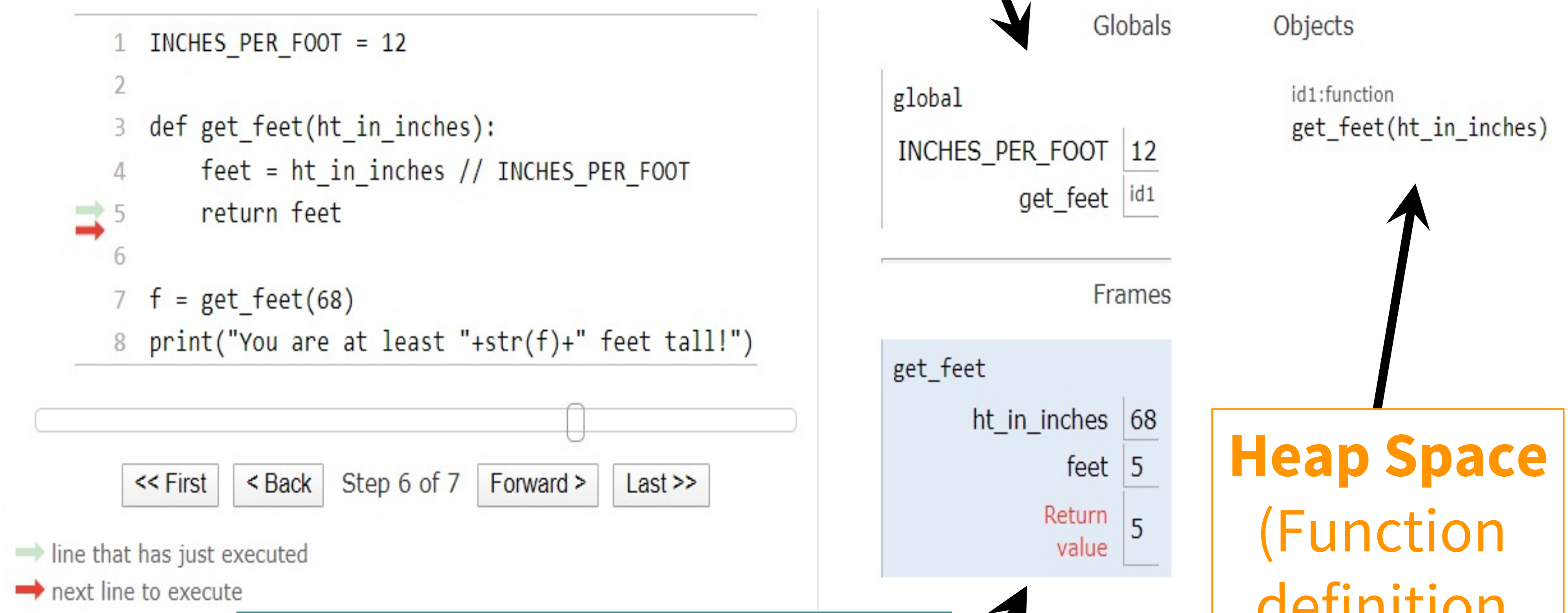


```
INCH_PER_FT = 12
def get_feet(ht_in_inches):
    return ht_in_inches // INCH_PER_FT
```



Function Definition vs. Call Frame

Global Space



```python
INCHES_PER_FOOT = 12

def get_feet(ht_in_inches):
    feet = ht_in_inches // INCHES_PER_FOOT
    return feet

f = get_feet(68)
print("You are at least "+str(f)+" feet tall!")
```

global

INCHES_PER_FOOT	12
get_feet	id1

get_feet

ht_in_inches	68
feet	5
Return value	5

id1:function
get_feet(ht_in_inches)

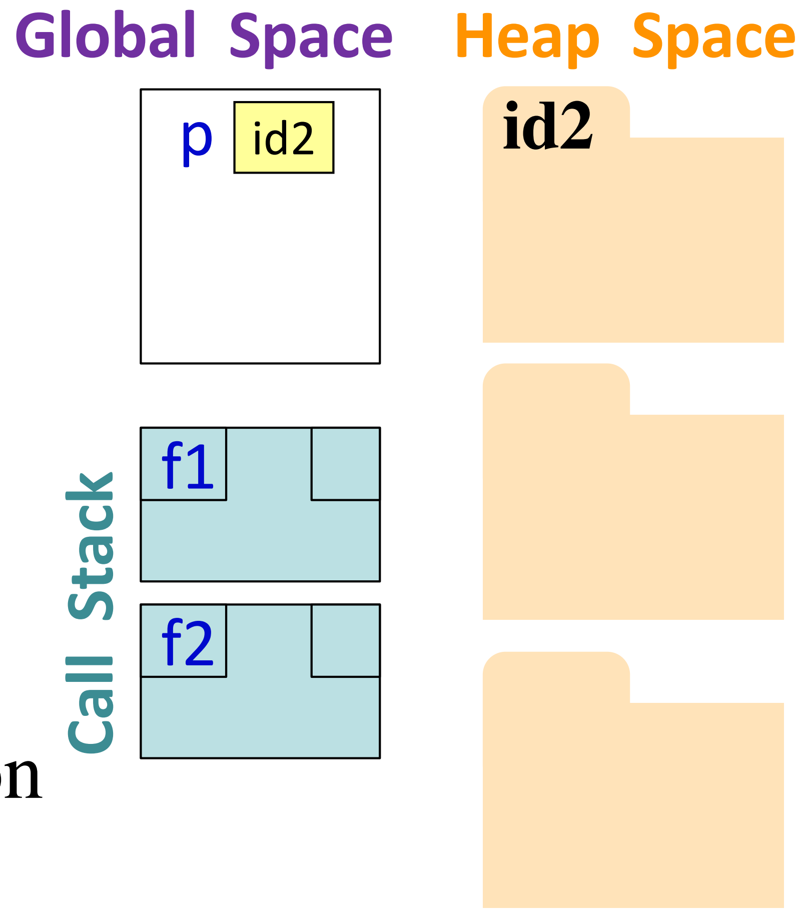
Heap Space (Function definition goes here)

Call Frame (memory for function **call**)

It’s alive!

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



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,

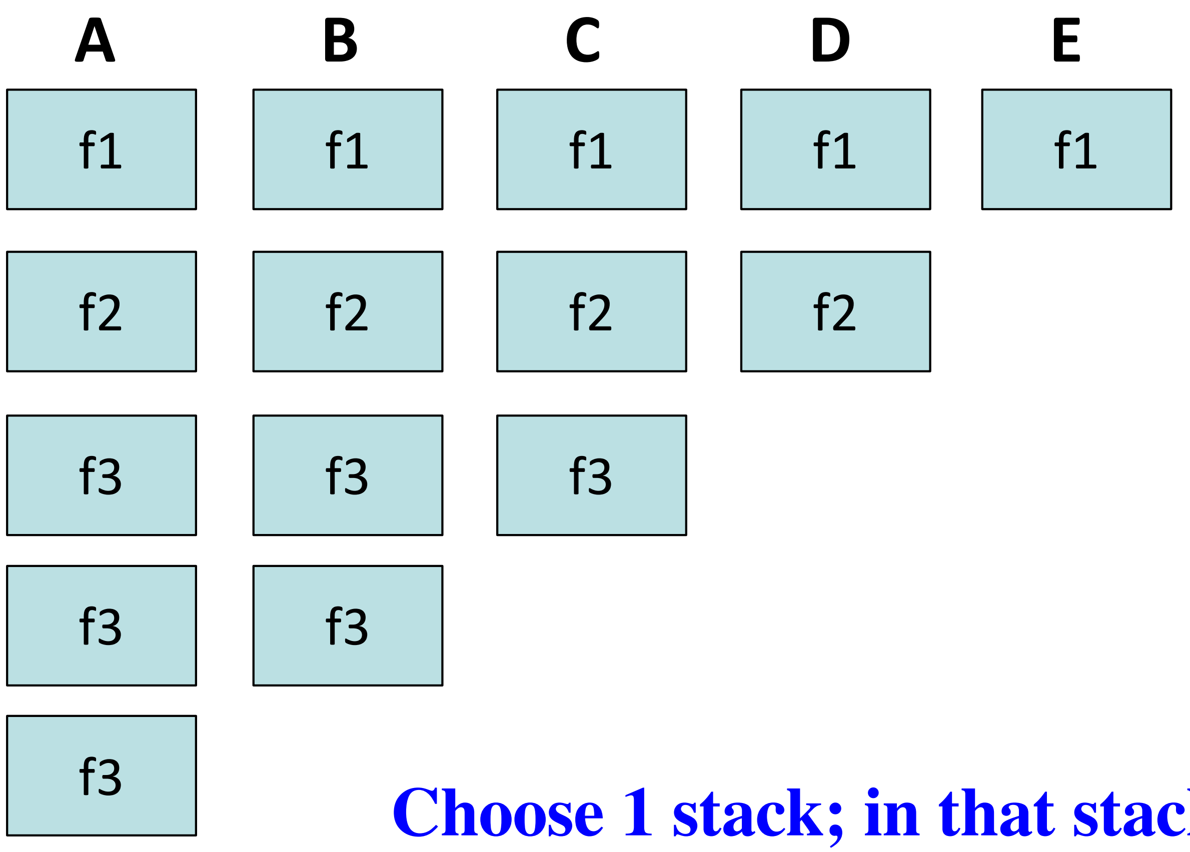
```
def get_feet(height_in_inches):
```

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?

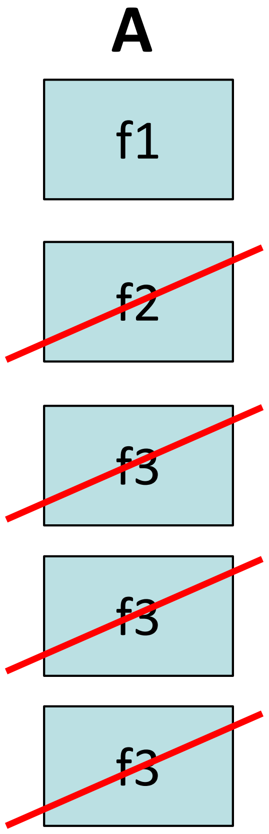
```
def f3():
    print("f3")

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

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

f1()
```



Choose 1 stack; in that stack cross out any frames that should have ended.

A3: 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()
```



Choose 1 stack; in that stack cross out any frames that should have ended.