http://www.cs.cornell.edu/courses/cs1110/2020sp

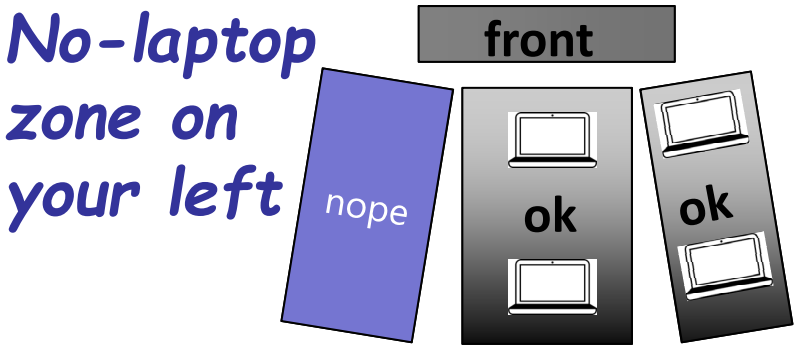
# Lecture 9: Memory in Python

CS 1110

Introduction to Computing Using Python

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

# Announcements




- You can use both **office hours** (held by profs and TAs) and **consulting hours** (held by undergrad consultants). See CS1110 office/consulting hours calendar on course website.
- Register your A1 group on CMS ***now***, before submitting any A1 file! Cannot form group after you submit.
- A1 first submission due Feb 19 Wedn at 11:59pm
- Read § 10.0-10.2, 10.4-10.6, 10.8-10.13 before next lecture

# Review: Conditionals and Testing

# Nested Conditionals

```
def what_to_wear(raining, freezing):
    if raining:
        if freezing:
            print("Wear a waterproof coat.")
        else:
            print("Bring an umbrella.")
    else:
        if freezing:
            print("Wear a warm coat!")
        else:
            print("A sweater will suffice.")
```

# Nested Conditionals

```
def what_to_wear(raining, freezing):
    if raining:
        if freezing:
            print("Wear a waterproof coat.")
        else:
            print("Bring an umbrella.")
    else:
        if freezing:
            print("Wear a warm coat!")
        else:
            print("A sweater will suffice.")
```

```
# an alternative
if _______:
    print( ... )
elif _______:
    print ( ... )
elif _______ :
    print ( ... )
else:
    print ( ... )
```

# Program Flow and Testing

Can use print statements to examine program flow

```
# Put max of x, y in z

if x > y:
    z = x
else:
    z = y
```

# Program Flow and Testing

Can use print statements to examine program flow

```
'before if'
'inside if x>y'
'after if'
```

x must have been greater than y

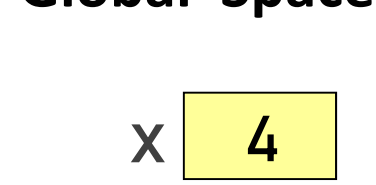
```
# Put max of x, y in z
print('before if')
if x > y:
    print('inside if x>y')
    z = x
else:
    print('inside else (x<=y)')
    z = y
print('after if')
```

"traces" or "breadcrumbs"

# Memory in Python

# Global Space

- **Global Space**
  - What you "start with"
  - Stores global variables
  - Lasts until you quit Python

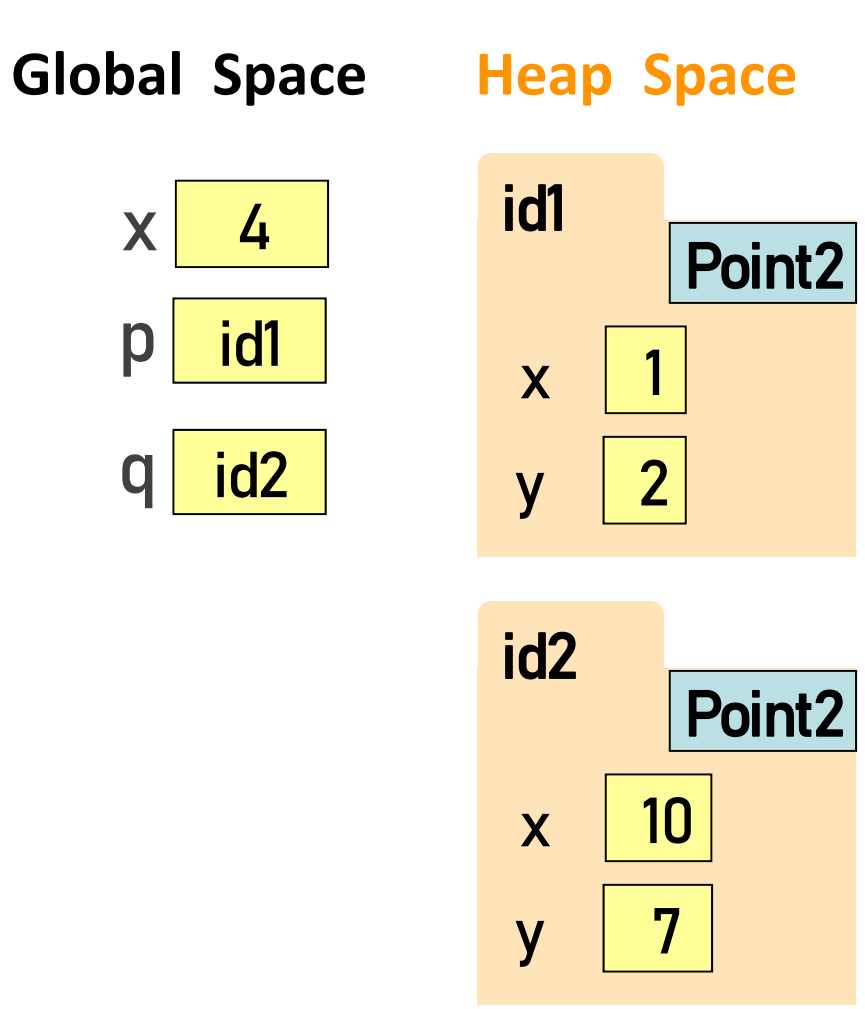



```
x = 4
```

# 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

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_coord(pt, n):
    pt.x = pt.x + n

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

1

**Global Space**

x 4

p id1

**Heap Space**

id1 Point2

x 1

y 2

**Call Frame**

adjust_x_coord 1

pt id1

n 4

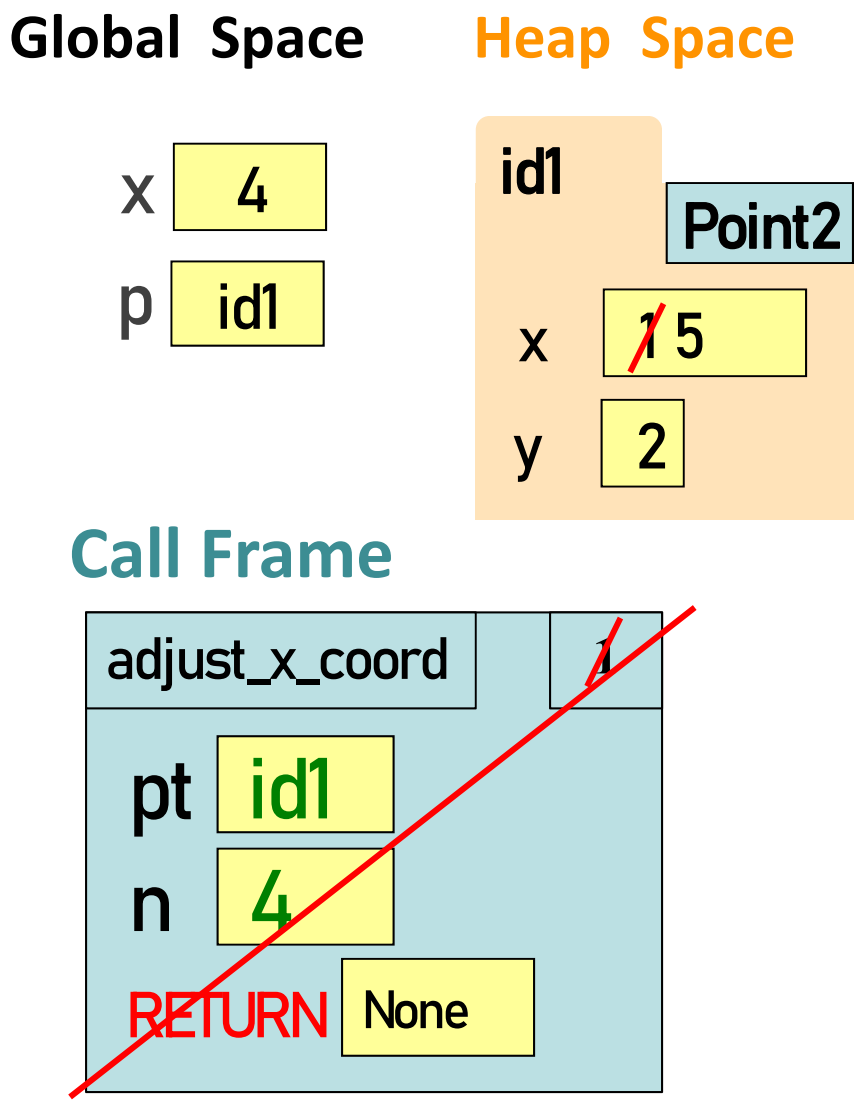
# Calling a Function Creates a Call Frame

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**

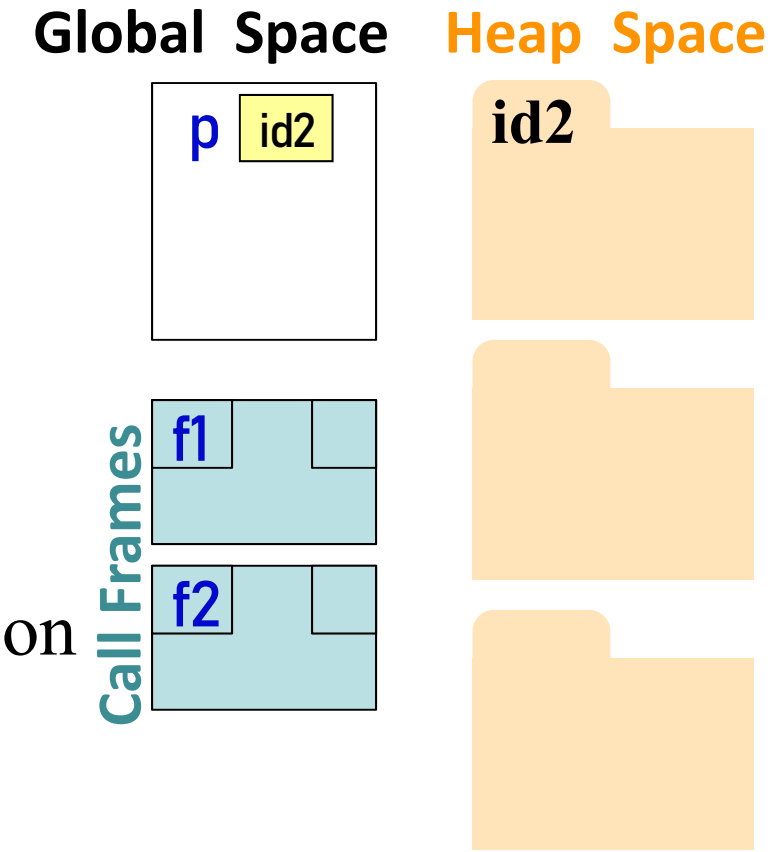
```
def adjust_x_coord(pt, n):
    pt.x = pt.x + n

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

1



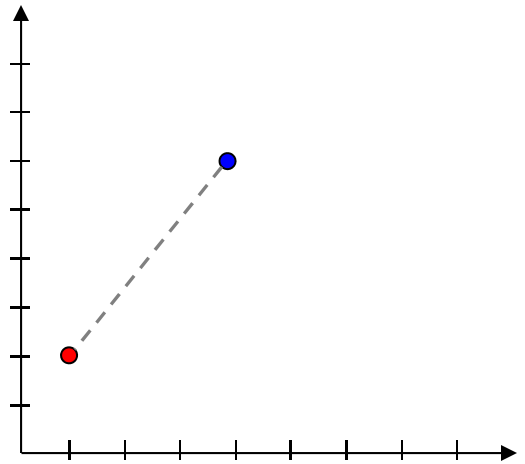
# Putting it all together

- **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

```
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 BAAAAD!

```
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!

```
def configure(pt, role):
    print("Where does the line " + role + "?")
    x = input("x: ")
    pt.x = int(x)
    y = input("y: ")
    pt.y = int(y)
    print("The line " +role+ "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

```
def configure(pt, role):
    print("Where does the line " + role + "?")
    x = input("x: ")
    pt.x = int(x)
    y = input("y: ")
    pt.y = int(y)
    print("The line " +role+ "s at ("+x+ ","+y+ ")." )

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

# Yay, Helper Functions!

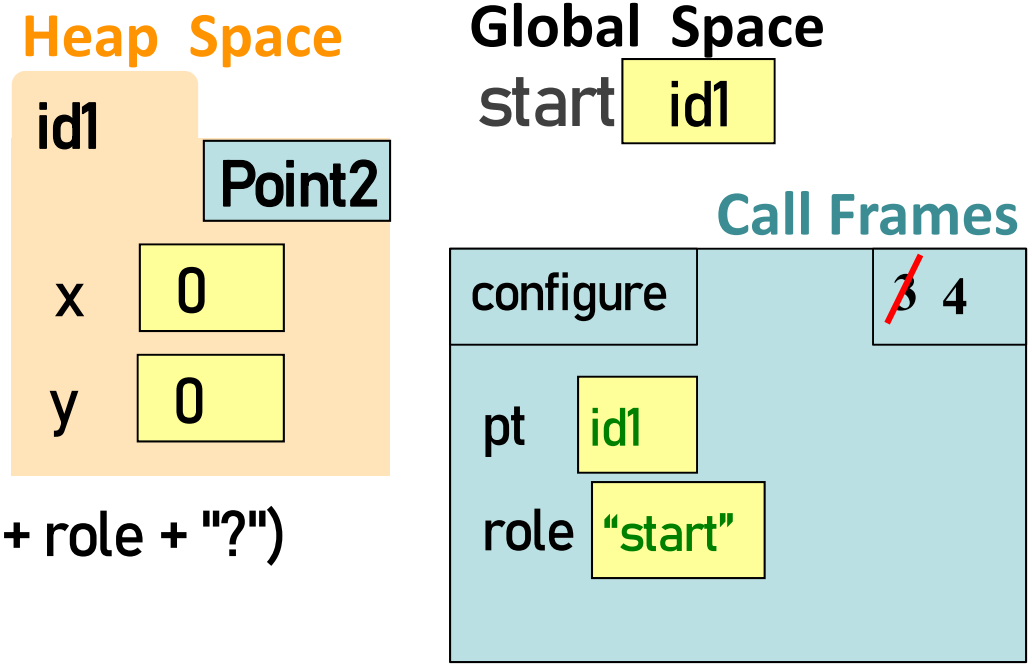
```
def get_coord(name):
    x = input(name+": ")
    return int(x)

def configure(pt, role):
    print("Where does the line " + role + "?")
    pt.x = get_coord("x")
    pt.y = get_coord("y")
    print("The line " +role+ "s at ("+x+ ","+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**

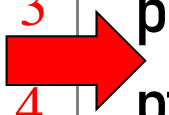
# Drawing Frames for Helper Functions (1)

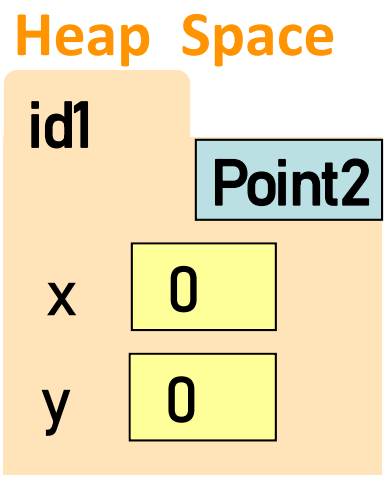
```
def get_coord(name):
1     x = input(name+": ")
2     return int(x)

def configure(pt, role):
3     print("Where does the line " + role + "?")
4     pt.x = get_coord("x")
5     pt.y = get_coord("y")
6     print("The line " +role+ "s at ("+str(pt.x)+
            ","+str(pt.y)+ ")." )
start = shape.Point2(0,0)
configure(start, "start")
```

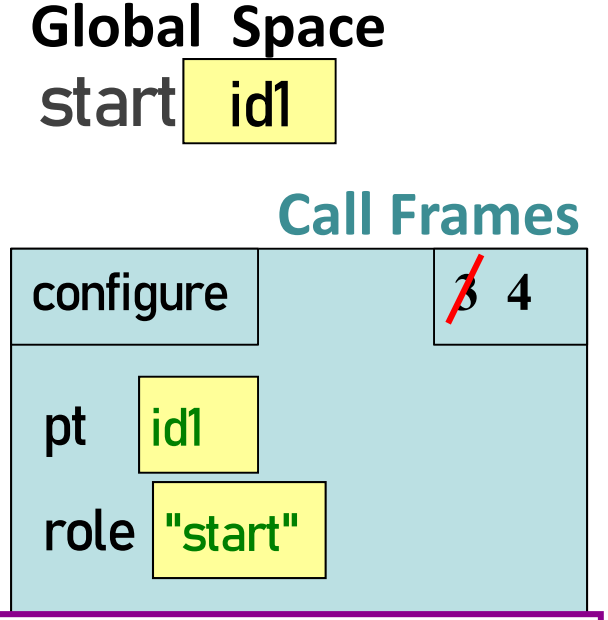


# Q: what do you do next?

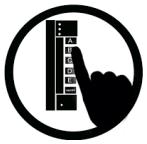
```
def get_coord(name):
1     x = input(name+": ")
2     return int(x)

def configure(pt, role):
3     print("Where does the line " + role + "?")
4     pt.x = get_coord("x")
5     pt.y = get_coord("y")
6     print("The line " +role+ "s at
            ","+str(pt.y)+ ")." )
start = shape.Point2(0,0)
configure(start, "start")
```





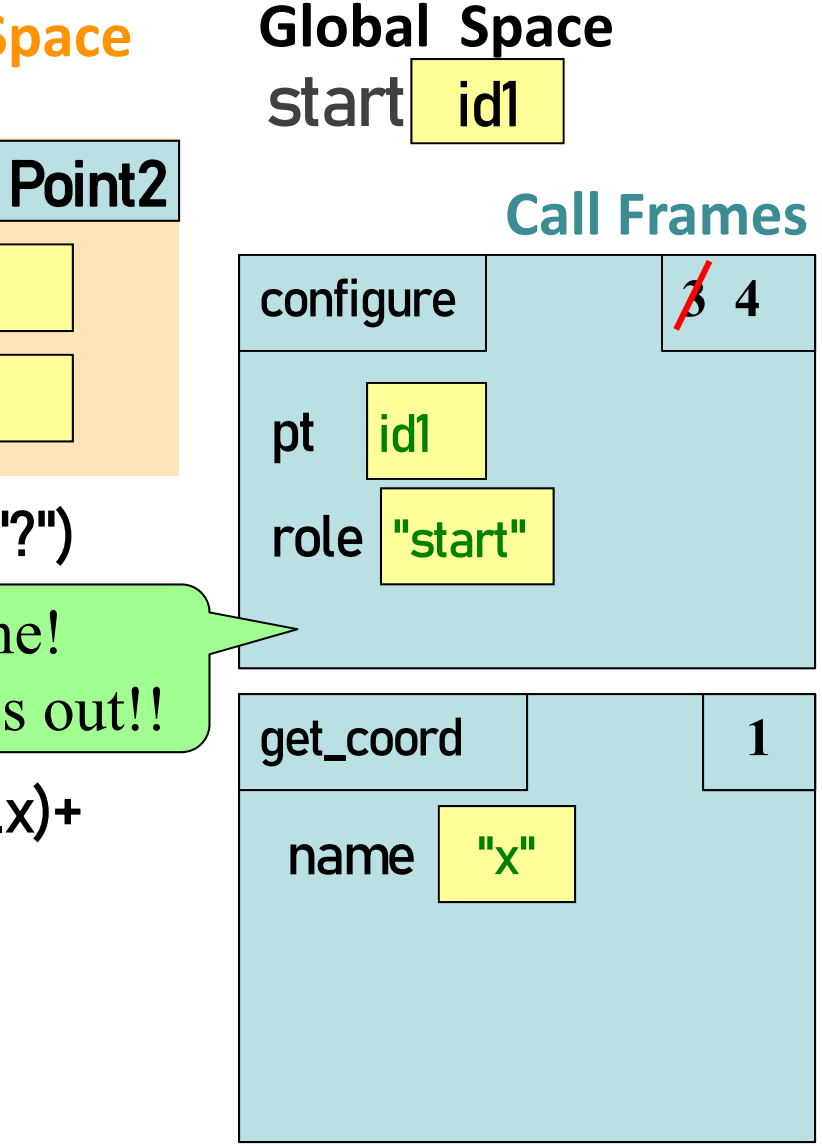




A: Cross out the **configure** call frame.
B: Create a **get_coord** call frame.
C: Cross out the 4 in the call frame.
D: A & B
E: B & C



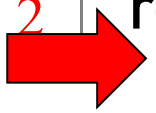
# Drawing Frames for Helper Functions (2)

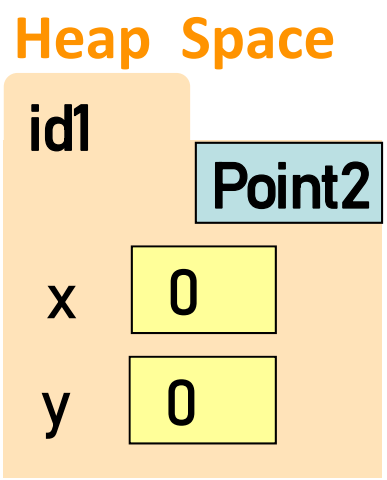
```
def get_coord(name):
1     x = input(name+": ")
2     return int(x)

def configure(pt, role):
3     print("Where does the line " + role + "?")
4     pt.x = get_coord("x")
5     pt.y = get_coord("y")
6     print("The line " +role+ "s at ("+str(pt.x)+
            ","+str(pt.y)+ ")." )
start = shape.Point2(0,0)
configure(start, "start")
```

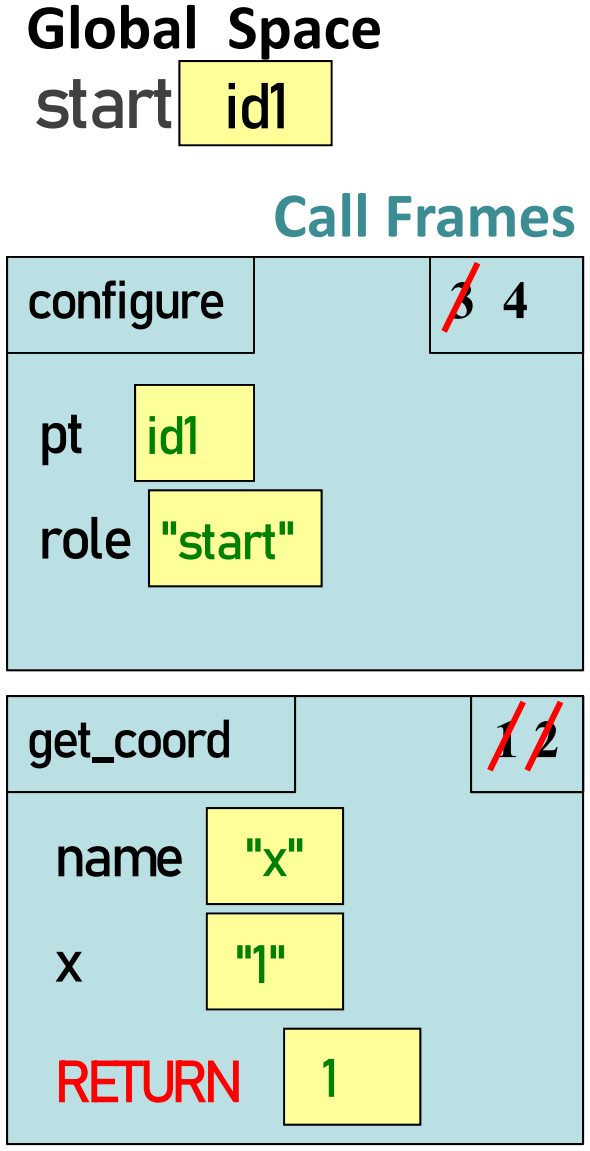


# Drawing Frames for Helper Functions (3)

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

def configure(pt, role):
3     print("Where does the line " + role + "?")
4     pt.x = get_coord("x")
5     pt.y = get_coord("y")
6     print("The line " +role+ "s at ("+str(pt.x)+
            ","+str(pt.y)+ ")." )
start = shape.Point2(0,0)
configure(start, "start")
```

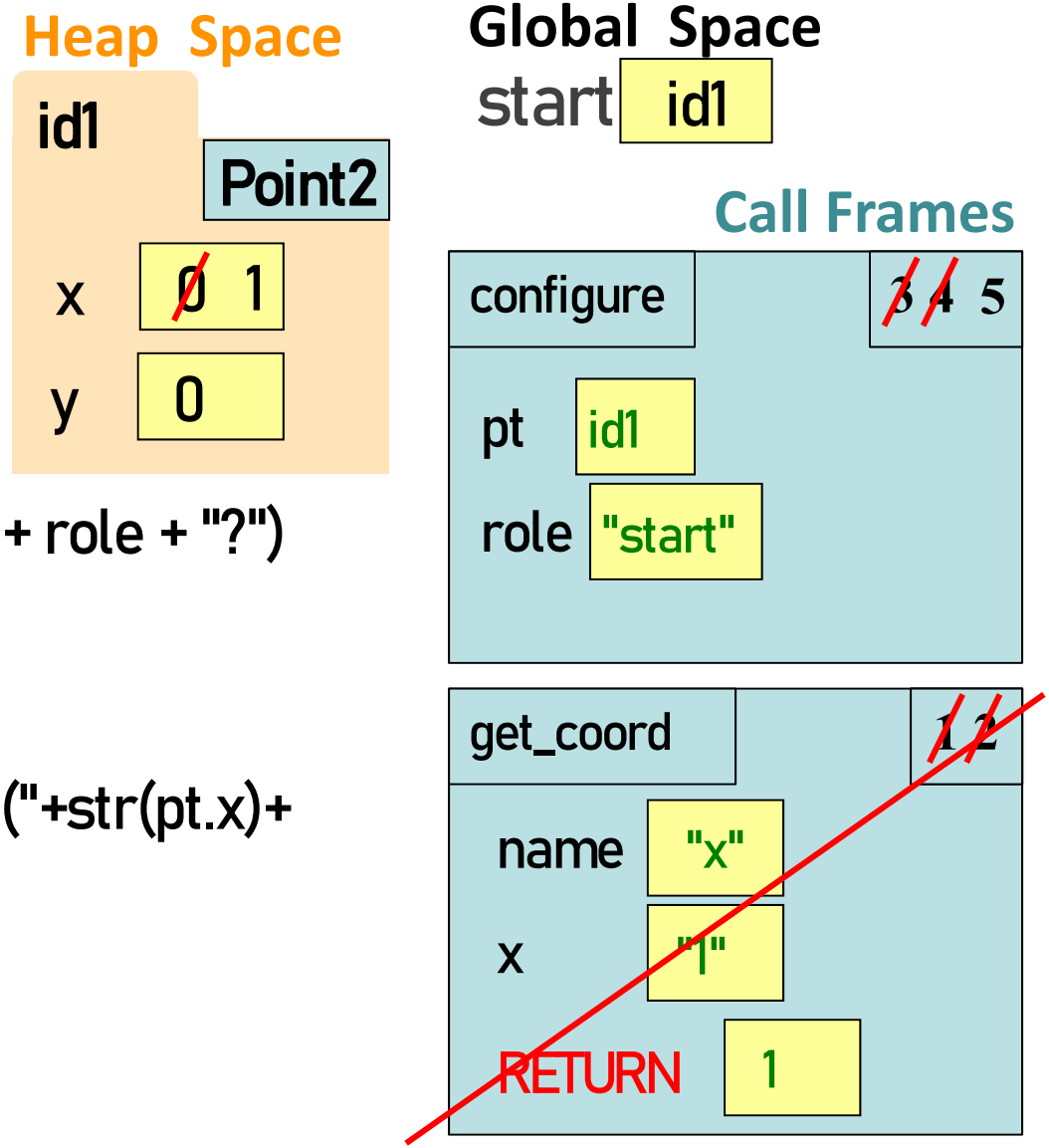




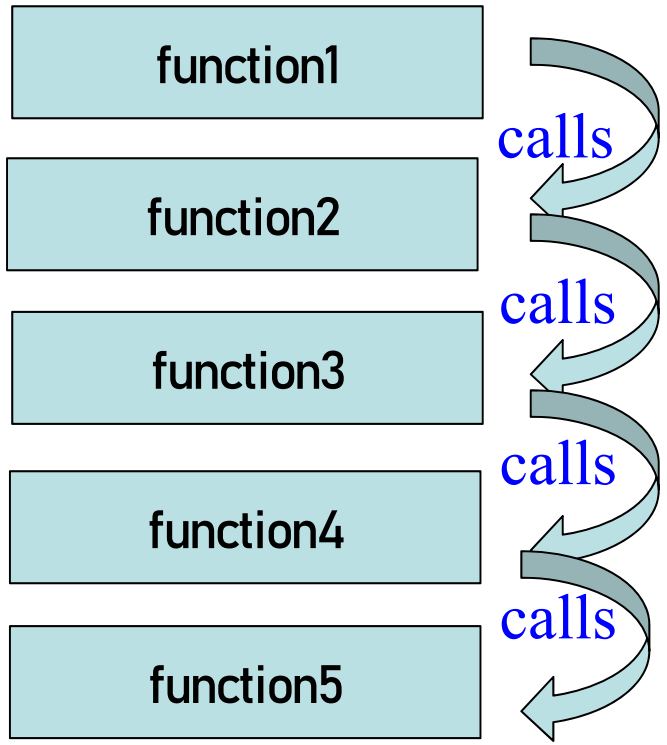
# Drawing Frames for Helper Functions (4)

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

def configure(pt, role):
3     print("Where does the line " + role + "?")
4     pt.x = get_coord("x")
5     pt.y = get_coord("y")
6     print("The line " +role+ "s at ("+str(pt.x)+
            ","+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):
1     x = input(name+": ")
2     return int(x1)

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

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

```
Where does the line start?
x: 1
Traceback (most recent call last):
  File "v3.py", line 15, in <module>
    configure(start, "start")
  File "v3.py", line 9, in configure
    pt.x = get_coord("x")
  File "v3.py", line 5, in get_coord
    return str(x1)
NameError: name 'x1' is not defined
```

# Modules and Global Space

## Import

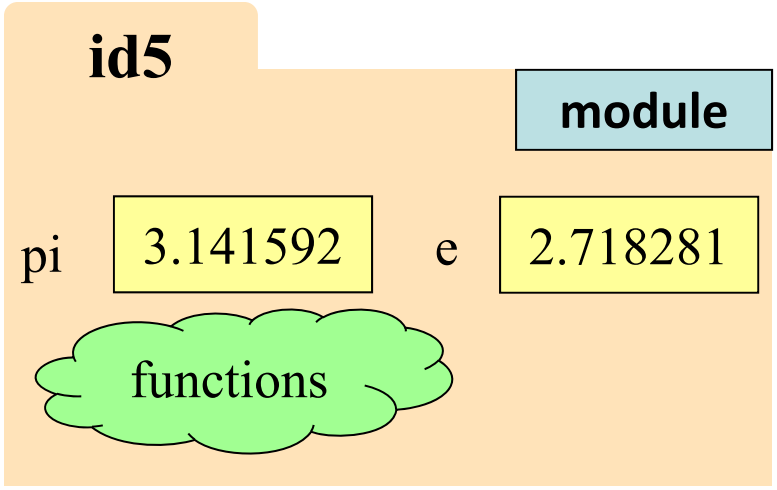
```
>>> import math
```

- 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**



# Modules vs Objects

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

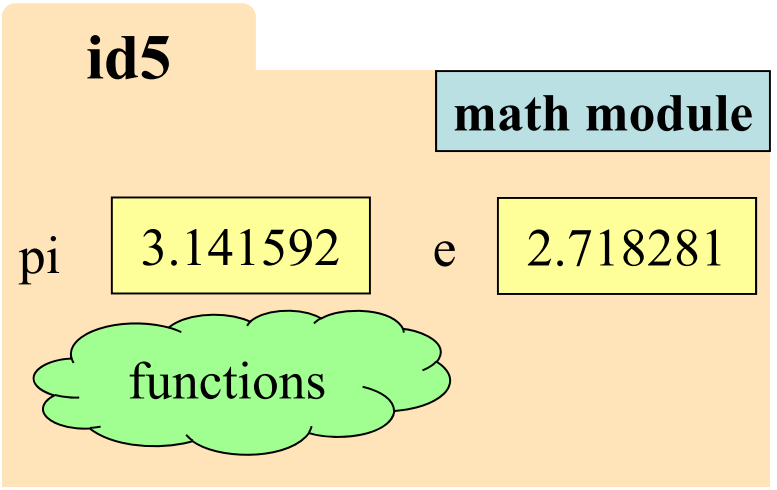
```
>>> p = shapes.Point3(5,2,3)
>>> p.x
```

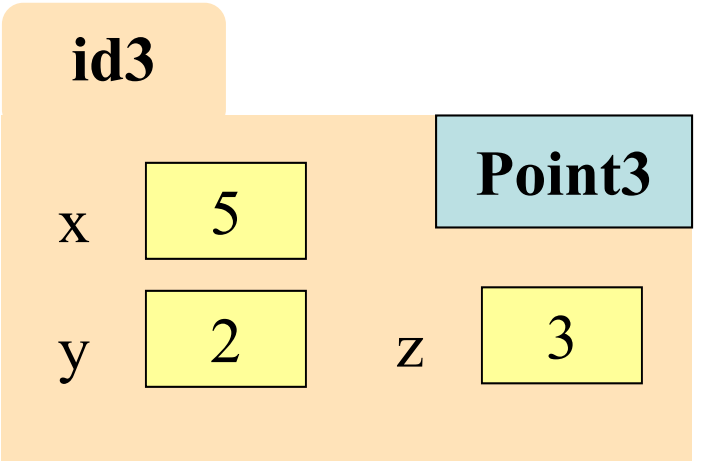
## Global Space

math | id5

p | id3

## Heap Space



# 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

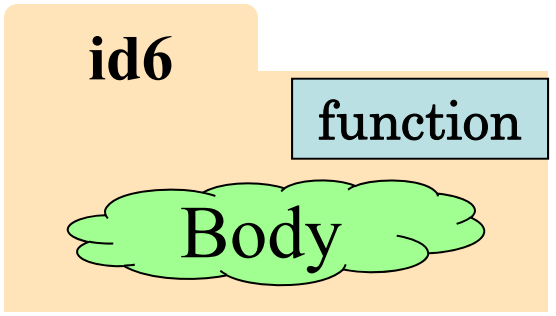
```
INCHES_PER_FT = 12
def get_feet(ht_in_inches):
    return ht_in_inches // INCHES_PER_FT
```

Body

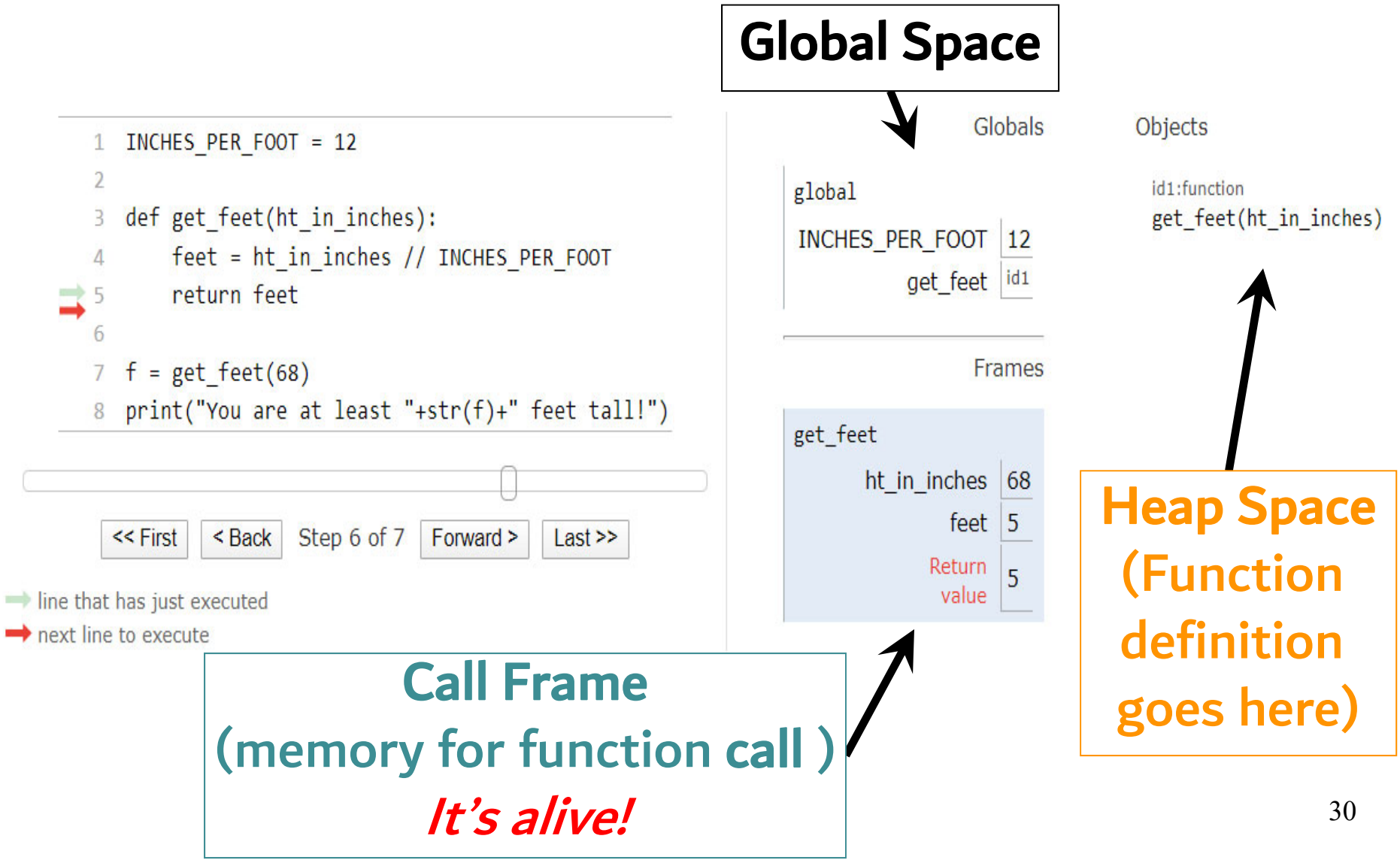
**Global Space**

| | |
|---|---|
| INCHES_PER_FT | 12 |
| get_feet | id6 |

**Heap Space**



# 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 Frame Stack**
  - Parameters
  - Other variables local to function
  - Lasts until function returns