

# CS 1110:

## Introduction to Computing Using Python

### Lecture 8

# Conditionals & Control Flow

[Andersen, Gries, Lee, Marschner, Van Loan, White]

# Announcements: Assignment 1

---

- Due *tonight* at 11:59pm.
  - Suggested early submit deadline of 2pm.
- Set CMS notifications to receive automatic emails when a grade is changed.
- First round of feedback should be out by Monday.
- If your A1 is not perfect, your first grade will be a 1.
  - This is a counter for how many times you have submitted.
  - It is not a permanent grade, can resubmit until March 2<sup>nd</sup>.
- Read section 2.3 of A1 carefully to understand how you can revise.

# Announcements

---

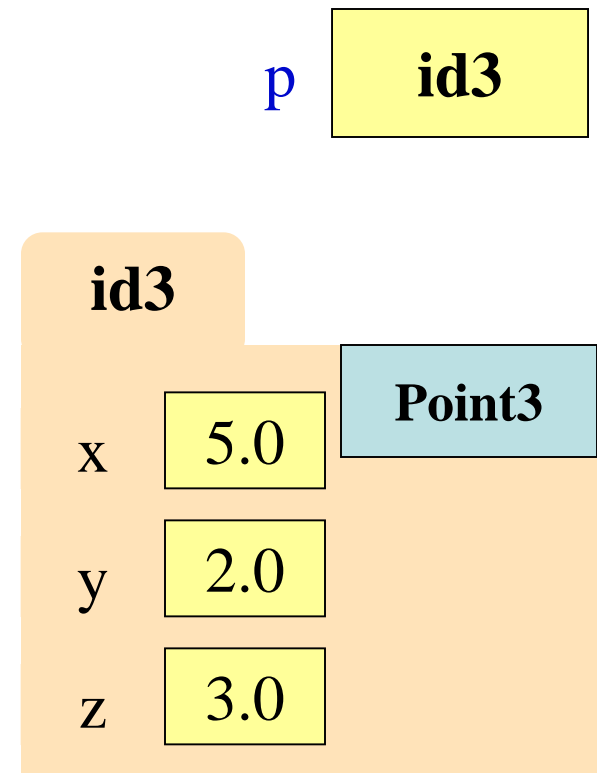
- Please do not post code to Piazza<sup>1</sup>
- Review the announcements from the end of Lecture 6 for policies:

<http://www.cs.cornell.edu/courses/cs1110/2017sp/lectures/02-14-17/presentation-06.pdf>

<sup>1</sup>actually violating academic integrity rules because you are showing code to others

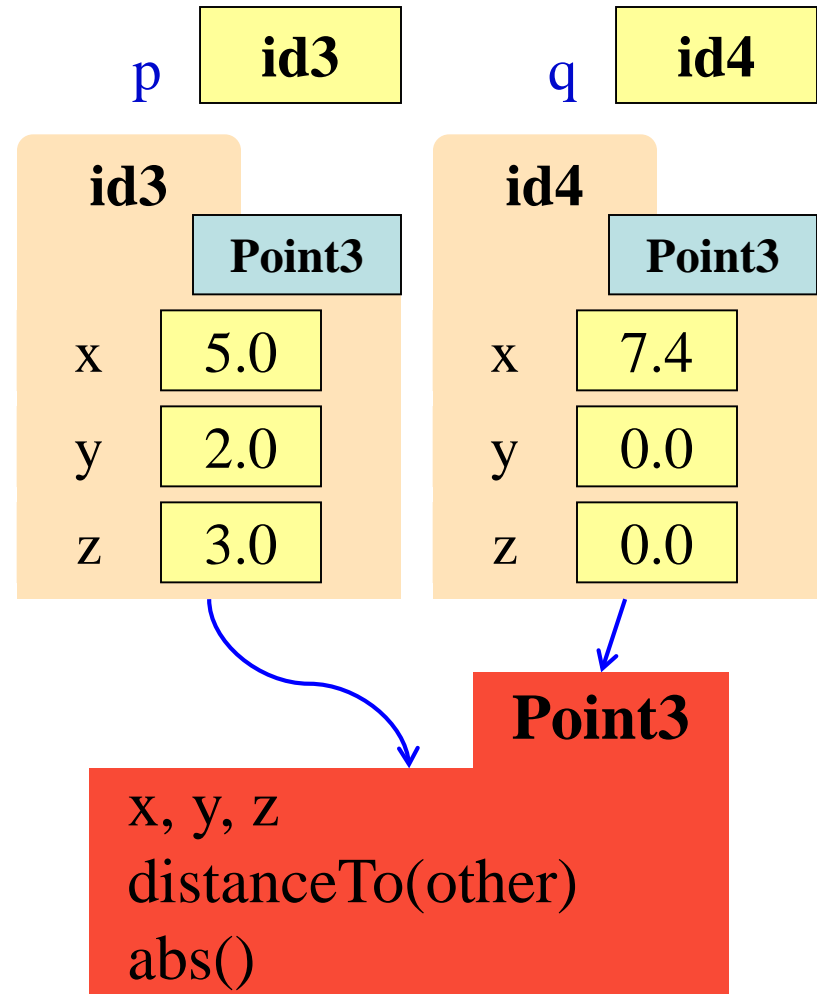
# Methods: Functions Tied to Classes

- **Method**: function tied to class
  - Method call looks like a function call preceded by a variable name:  
*<variable>.<method>(<arguments>)*
  - **Example**: `p.distanceTo(q)`
- Just like we saw for strings
  - `s = 'abracadabra'`
  - `s.index('a')`
- Are strings objects? **Actually, yes.**



# Name Resolution

- $\langle object \rangle . \langle name \rangle$  means
  - Go the folder for *object*
  - Look for attribute/method *name*
  - If missing, check *class folder*
- Class folder is a **shared folder**
  - Only one for the whole class
  - Shared by all objects of class
  - Stores common features
  - Typically where methods are



# Structure vs. Flow

---

## Program Structure

---

- Order in which statements **are written** in scripts and modules
- Not necessarily the order in which Python executes them

## Control Flow

---

- Order in which statements are **actually executed** at runtime
  - Statements may be:
    - skipped
    - executed more than once

Have already seen this  
difference with functions

# Structure vs. Flow: Example

---

## Program Structure

---

```
def foo():  
    print 'Hello'
```

This statement  
listed only once

```
# Script Code
```

```
foo()
```

```
foo()
```

```
foo()
```

## Control Flow

---

```
C:\> python foo.py
```

```
'Hello'
```

```
'Hello'
```

```
'Hello'
```

Statement  
executed 3 times

# Conditionals: If-Statements

---

## Format

```
if <boolean-expression>:  
    <statement>  
    ...  
    <statement>
```

## Example

```
# Put x in z if it is positive  
if x > 0:  
    z = x
```

### Execution:

if *<Boolean-expression>* is true, then execute all of the statements indented directly underneath (until first non-indented statement)



# What gets printed?

---

a = 0

prints 0

print a

# What gets printed?

---

a = 0

a = a + 1

prints 1

print a

# What gets printed?

---

```
a = 0
```

```
if a == 0:
```

```
    a = a + 1
```

prints 1

```
print a
```

# What gets printed?

---

```
a = 0
```

```
if a == 1:
```

```
    a = a + 1
```

prints 0

```
print a
```

# What gets printed?

---

```
a = 0
```

```
if a == 1:
```

```
    | a = a + 1
```

```
a = a + 1
```

```
print a
```

prints 1

# What gets printed?

---

```
a = 0
```

```
if a == 0:
```

```
    | a = a + 1
```

```
a = a + 1
```

```
print a
```

prints 2

# What gets printed?

a = 0

Executed

if a == 0:

| a = a + 1

Executed

if a == 0:

| a = a + 1

Skipped

a = a + 1

Executed

print a

A: 0

B: 1

C: 2 **CORRECT**

D: 3

E: I do not know

# Conditionals: If-Else-Statements

---

## Format

```
if <boolean-expression>:  
    <statement>  
    ...  
else:  
    <statement>  
    ...
```

## Example

```
# Put max of x and y in z  
if x > y:  
    z = x  
else:  
    z = y
```

### Execution:

if *<Boolean-expression>* is true, then execute statements indented under **if**; otherwise execute the statements indented under **else**

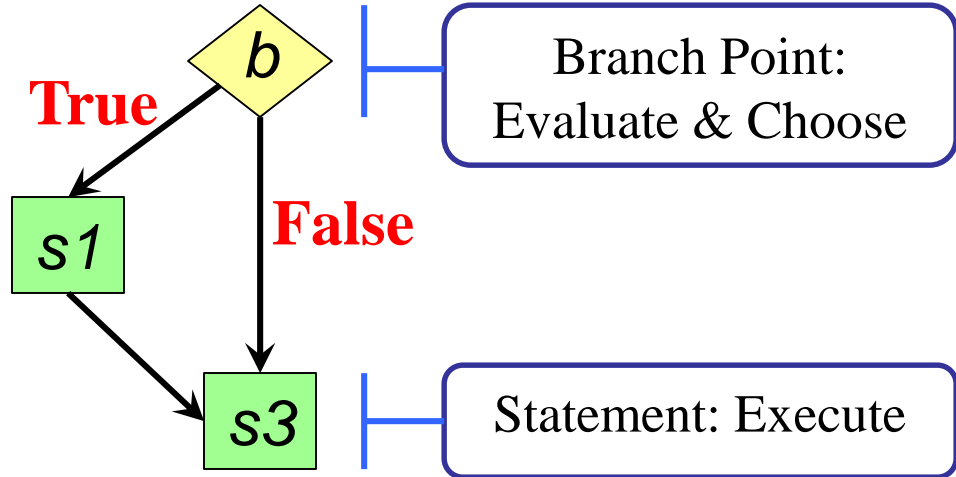


# Conditionals: “Control Flow” Statements

**if**  $b$  :

|  $s1$  # statement

|  $s3$



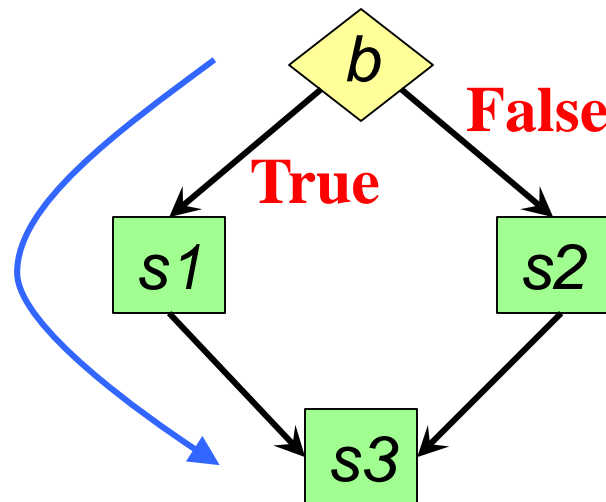
**if**  $b$  :

|  $s1$

**else:**

|  $s2$

|  $s3$



**Flow**

Program only takes one path each execution

# What gets printed?

---

```
a = 0
```

```
if a == 0:
```

```
| a = a + 1
```

prints 1

```
else:
```

```
| a = a + 1
```

```
print a
```

# What gets printed?

---

```
a = 0
```

```
if a == 1:
```

```
| a = a + 1
```

prints 1

```
else:
```

```
| a = a + 1
```

```
print a
```

# What gets printed?

---

```
a = 0
```

```
if a == 1:
```

```
| a = a + 1
```

prints 2

```
else:
```

```
| a = a + 1
```

```
a = a + 1
```

```
print a
```

# What gets printed?

---

```
a = 0
```

```
if a == 1:
```

```
    a = a + 1
```

```
else:
```

```
    a = a + 1
```

```
    a = a + 1
```

```
a = a + 1
```

```
print a
```

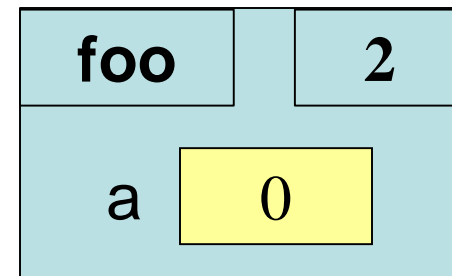
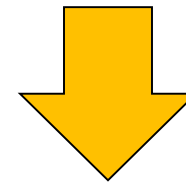
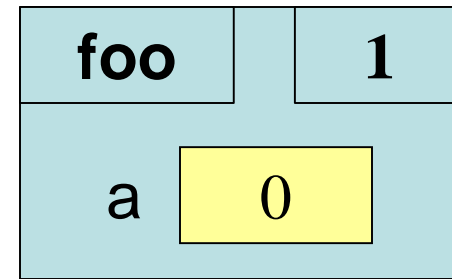
prints 3

# Program Flow and Call Frames

- **if** can change which statement is executed next

```
def foo(a):  
1 |   if a == 0:  
2 |     print "hi"  
3 |   print "bye"
```

foo(0)

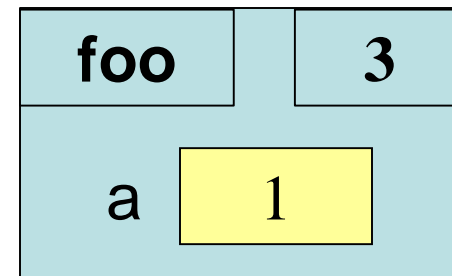
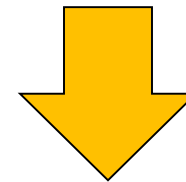
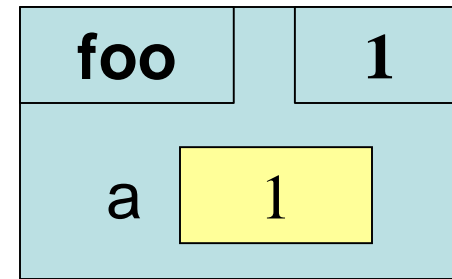


# Program Flow and Call Frames

- **if** can change which statement is executed next

```
def foo(a):  
1 |   if a == 0:  
2 |     print "hi"  
3 |   print "bye"
```

foo(1)



# Program Flow and Call Frames

---

```
def max(x,y):
```

```
1 | if x > y:  
2 |     return x  
3 | return y
```

```
max(0,3):
```

max		1
x	0	
y	3	

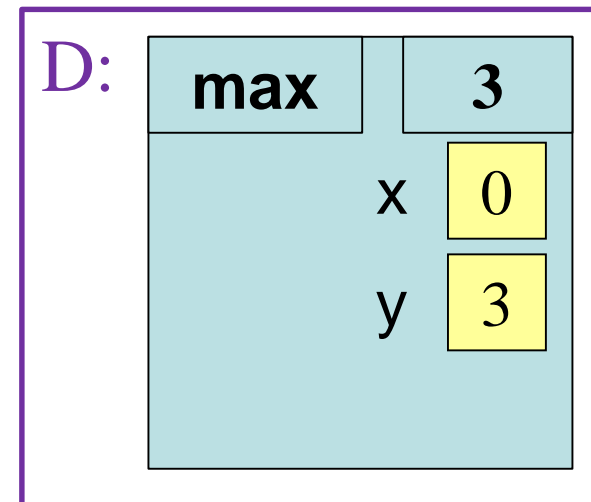
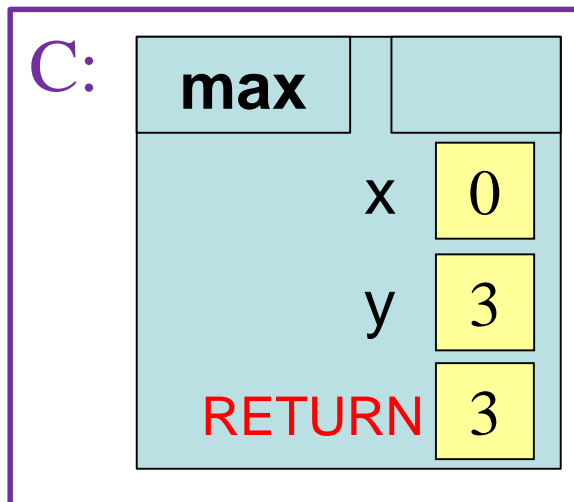
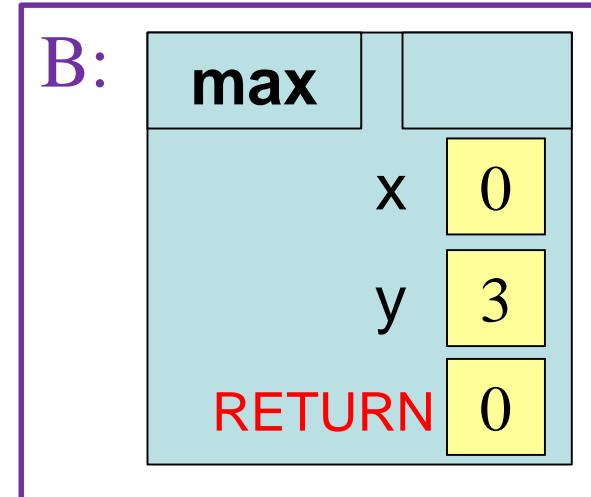
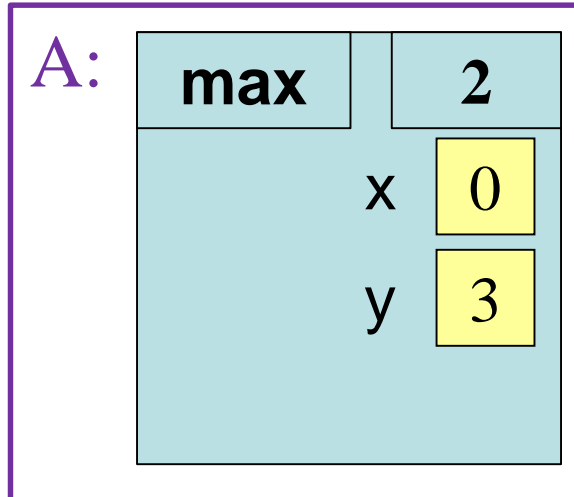
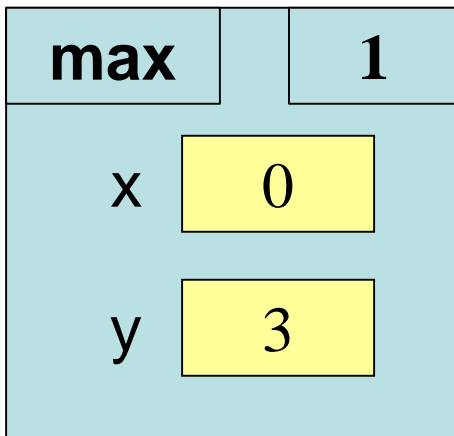


# What happens next?

```
def max(x,y):
```

```
1 | if x > y:  
2 |     return x  
3 | return y
```

Current call frame:



# Program Flow and Call Frames

```
def max(x,y):
```

```
1 | if x > y:  
2 |     return x  
3 | return y
```

```
max(0,3):
```

max		3
x	0	
y	3	

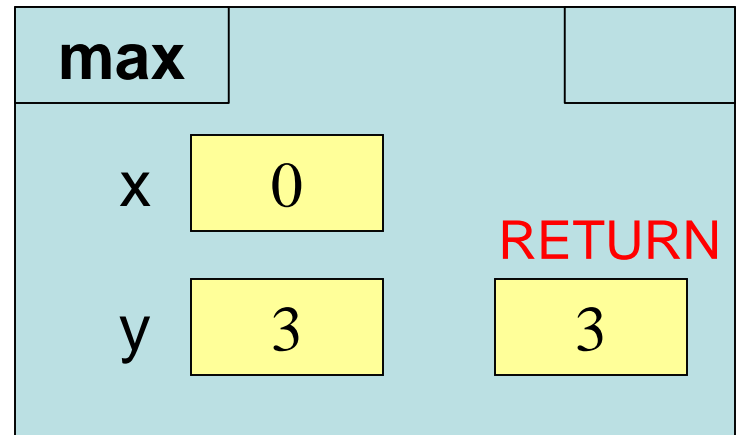
Skips line 2

# Program Flow and Call Frames

```
def max(x,y):
```

```
1 | if x > y:  
2 |     return x  
3 | return y
```

```
max(0,3):
```



Skips line 2

# Program Flow and Variables

---

- Variables continue to exist outside of **if**

```
a = 0
```

```
if a == 0:
```

```
    a = a + 1
```

```
print a
```

- Also continue to exist even if *created* in **if**

# Program Flow and Variables

---

```
a = 0
```

```
if a == 0:
```

```
|   b = 0
```

```
print b
```



prints 0

# Program Flow and Variables

---

```
a = 0
```

```
if a == 1:
```

```
|   b = 0
```

```
print b
```

**Error!**

# Program Flow and Variables

---

```
def zero_or_one(b):
```

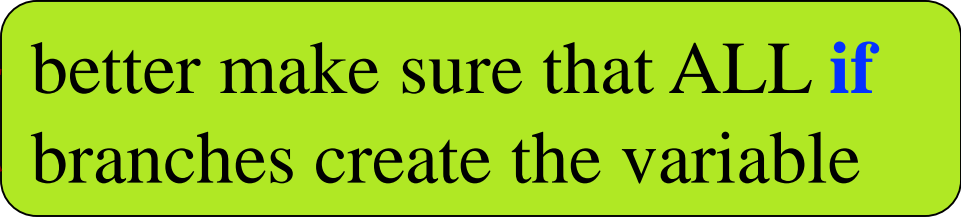
```
    if b:
```

```
        a = 0
```

```
    else:
```

```
        a = 1
```

```
    print a
```



better make sure that ALL **if** branches create the variable

# Program Flow and Testing

- Can use print statements to examine program flow

'before if'

'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 'if x>y'
```

```
    z = x
```

```
else:
```

```
    print 'else x<=y'
```

```
    z = y
```

```
print 'after if'
```

Traces



# Conditionals: If-Elif-Else-Statements

---

## Format

```
if <Boolean expression>:  
|   <statement>  
|   ...  
elif <Boolean expression>:  
|   <statement>  
|   ...  
...  
else:  
|   <statement>  
|   ...
```

## Example

```
# Put max of x, y, z in w  
if x > y and x > z:  
|   w = x  
elif y > z:  
|   w = y  
else:  
|   w = z
```

# Conditionals: If-Elif-Else-Statements

---

## Format

```
if <Boolean expression>:  
    <statement>  
    ...  
elif <Boolean expression>:  
    <statement>  
    ...  
...  
else:  
    <statement>  
    ...
```

## Notes on Use

- No limit on number of **elif**
  - Must be between **if**, **else**
- **else** is optional
  - **if-elif** by itself is fine
- Booleans checked in order
  - Once Python finds a true *<Boolean-expression>*, skips over all the others
  - **else** means **all** are false

# If-Elif-Else

---

```
a = 2
```

```
if a == 2:
```

```
    a = 3
```

```
elif a == 3:
```

```
    a = 4
```

```
print a
```

What gets printed?

A: 2

B: 3 **CORRECT**

C: 4

D: I do not know

# If-Elif-Else

---

a = 2

**if** a == 2:

    a = 3

**elif** a == 3:

    a = 4

**print** a

prints 3

a = 2

**if** a == 2:

    a = 3

**if** a == 3:

    a = 4

**print** a

prints 4

# Nested Conditionals

---

```
def test_for_zeros(a, b):  
    if a == 0:  
        if b == 0:  
            print "Both arguments are zero"  
        else:  
            print "The first argument is zero"  
    else:  
        if b == 0:  
            print "The second argument is zero"  
        else:  
            print "Neither argument is zero"
```

# Conditional Expressions

---

## Format

---

$e_1$  **if**  $bexp$  **else**  $e_2$

- $e_1$  and  $e_2$  are any expression
- $bexp$  is a Boolean expression
- This is an expression!

## Example

---

# Put max of x, y in z

$z = x$  **if**  $x > y$  **else**  $y$



expression,  
not statement