Announcements

- Please do not post code to Piazza[1]
- 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 2nd.
- 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

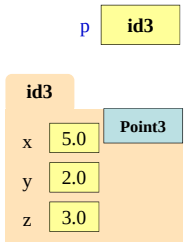
- Set your CMS notifications to get email when a grade is changed. The first round of feedback should be out by Monday.
- Read section 2.3 of A1 carefully to understand how you can revise.

[1]actually violating academic integrity rules because you are showing code to others

Methods: Functions Tied to Classes

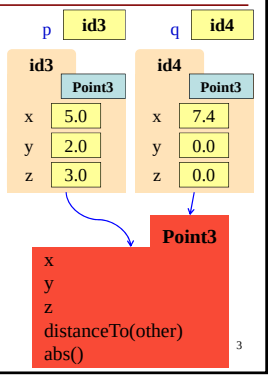
- **Method**: function tied to object
  - Method call looks like a function call preceded by a variable name:

    ⟨*variable*⟩.⟨*method*⟩(⟨*arguments*⟩)
  - **Example**: p.distanceTo(q)
  - **Example**: p.abs() # makes x,y,z ≥ 0
- Just like we saw for strings
  - s = 'abracadabra'
  - s.index('a')
- Are strings objects? Actually, **yes**. But this is not so important.



Name Resolution

- ⟨*object*⟩.⟨*name*⟩ means
  - Go the folder for *object*
  - Look for attr/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

Program 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'

# Script Code
foo()
foo()
foo()
```

Statement listed once

Program Flow

```
>>> python foo.py
'Hello'
'Hello'
'Hello'
```

Statement executed 3x

Bugs can occur when we get a flow other than one that we where expecting

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 statements indented directly underneath (until first non-indented statement)

Conditionals: If-Else-Statements

Format	Example

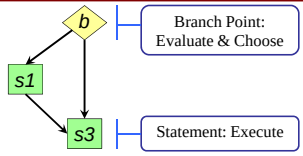
if <*boolean-expression*>:
  <*statement*>
  …
else:
  <*statement*>
  …
	# Put max of x, y in z
	if x > y:
	  z = x
	else:
	  z = y

Execution:

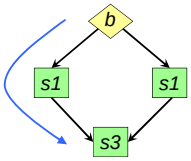
if <*Boolean expression*> is true, then execute statements indented under if; otherwise execute the statements indented under else

Conditionals: "Control Flow" Statements

if *b* :
  s1 # statement
s3



if *b* :
  s1
else:
  s2
s3



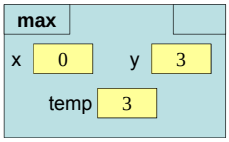
Flow
Program only takes one path each execution

Program Flow vs. Local Variables

```
def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp

    return y
```

- temp is needed for swap
  - x = y loses value of x
  - "Scratch computation"
  - Primary role of local vars
- max(3,0):



Program Flow and Testing

- Must understand which flow caused the error
  - Unit test produces error
  - Use print statements to examine program flow

```
# 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	Example

if <*boolean-expression*>:
  <*statement*>
  …
elif <*boolean-expression*>:
  <*statement*>
  …
…
else:
  <*statement*>
  …
	# Put max of x, y, z in w
	if x > y and x > z:
	  w = x
	elif y > z:
	  w = y
	else:
	  w = z