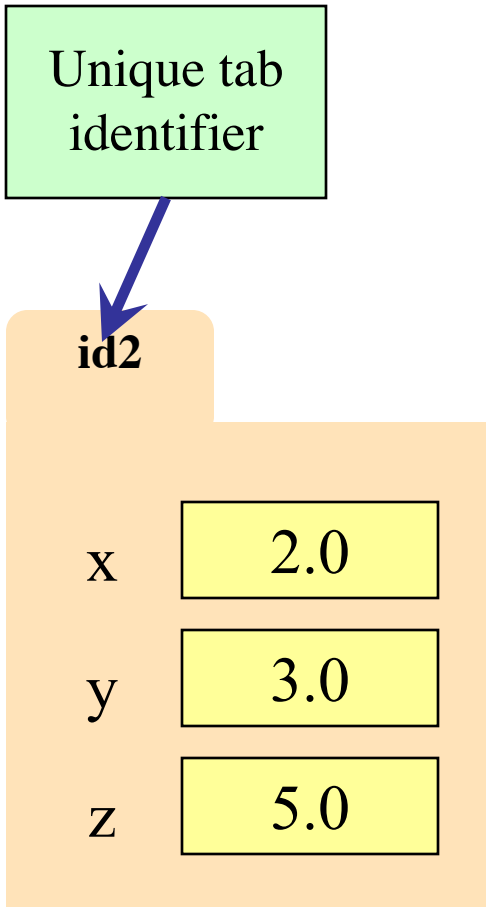
Mini-Lecture 18

Classes

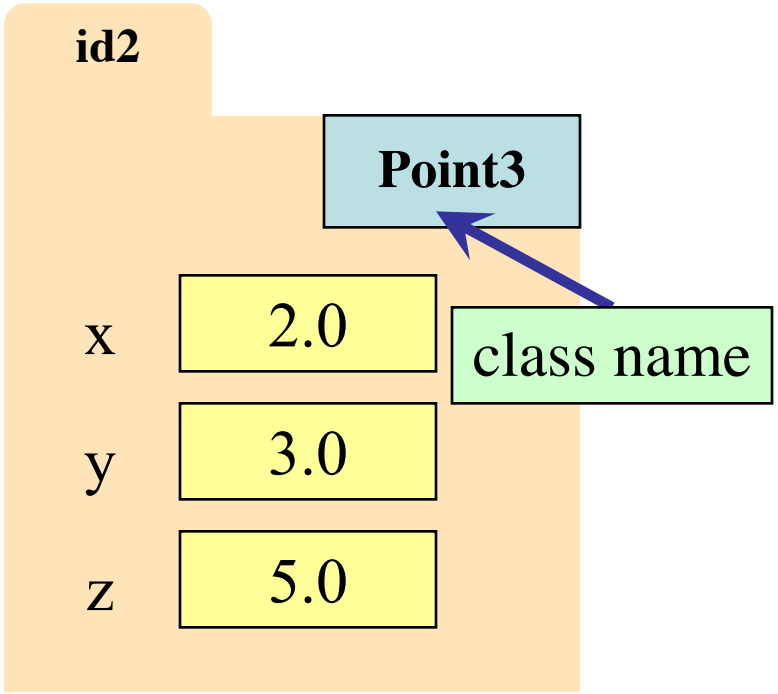
Recall: Objects as Data in Folders

- An object is like a **manila folder**
- It contains other variables
  - Variables are called **attributes**
  - Can change values of an attribute (with assignment statements)
- It has a "tab" that identifies it
  - Unique number assigned by Python
  - Fixed for lifetime of the object

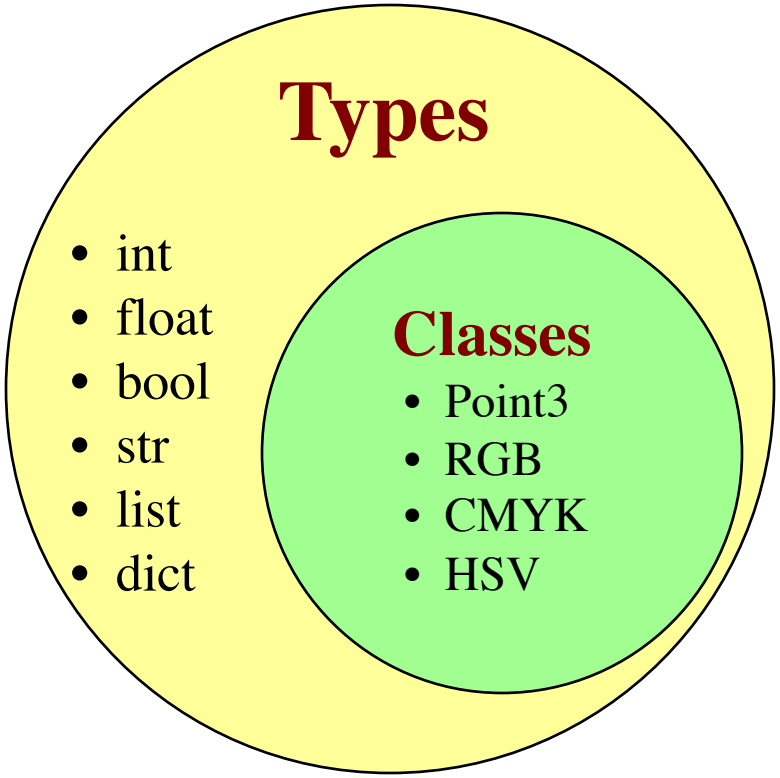


Recall: Classes are Types for Objects

- Values must have a type
  - An object is a **value**
  - Object type is a **class**



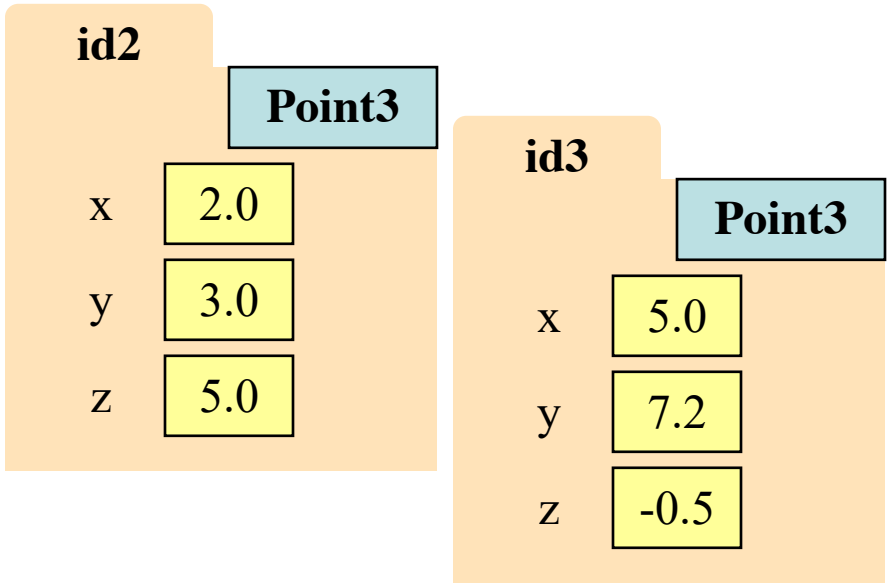
- Classes are any type not already built-into Python



Classes Have Folders Too

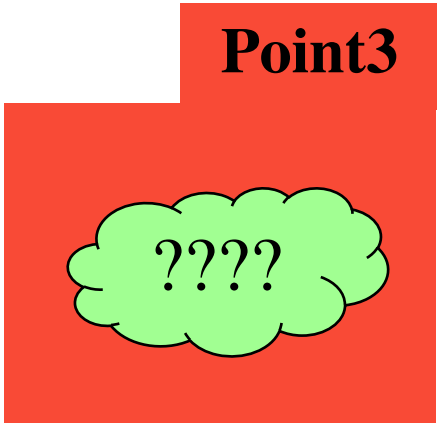
Object Folders

- Separate for each *instance*



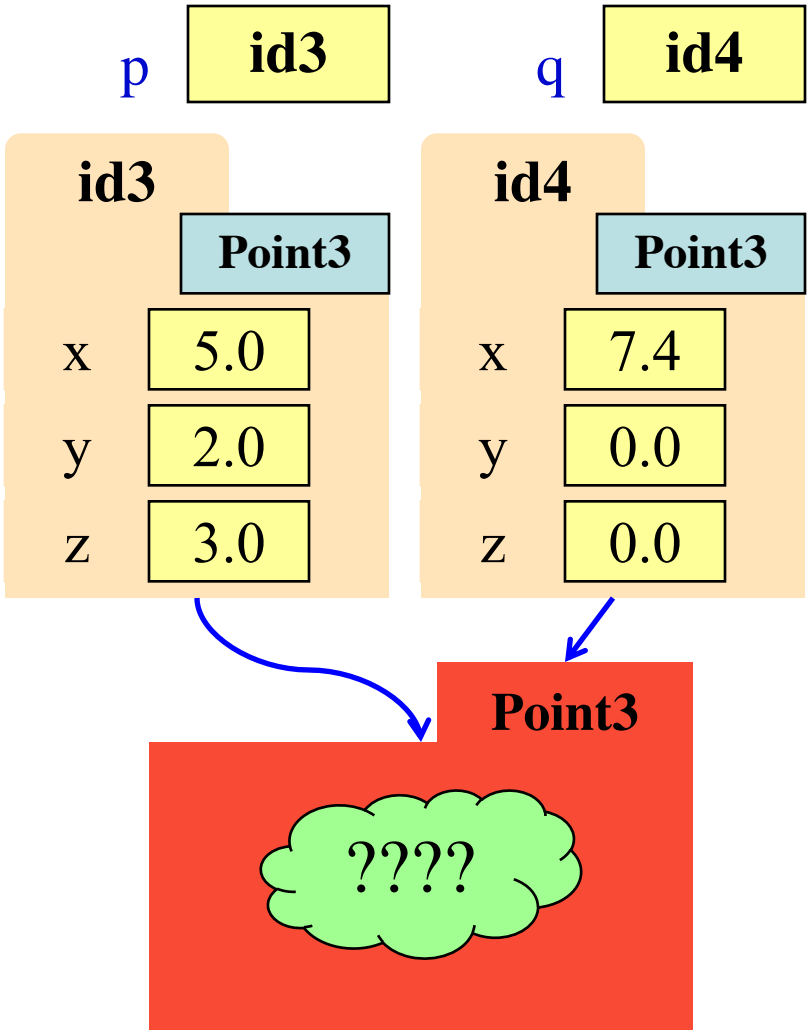
Class Folders

- Data common to all instances



Name Resolution for Objects

- ⟨*object*⟩.⟨*name*⟩ means
  - Go the folder for *object*
  - Find attribute/method *name*
  - If missing, check **class folder**
  - If not in either, raise error
- What is in the class folder?
  - Data common to **all** objects
  - First must understand the ***class definition***



The Class Definition

Goes inside a module, just like a function definition.

class <*class-name*>(object):

"""Class specification"""

<*function definitions*>

<*assignment statements*>

<*any other statements also allowed*>

Example

```
class Example(object):
    """The simplest possible class."""
    pass
```

The Class Definition

Goes inside a module, just like a function definition.

keyword class
Beginning of a class definition

class <*class-name*>(object):

Do not forget the colon!

more on this later

Specification (similar to one for a function)

"""Class specification"""

<*function definitions*>

to define **methods**

<*assignment statements*>

…but not often used

to define **attributes**

<*any other statements also allowed*>

```
class Example(object):
    """The simplest possible class."""
    pass
```

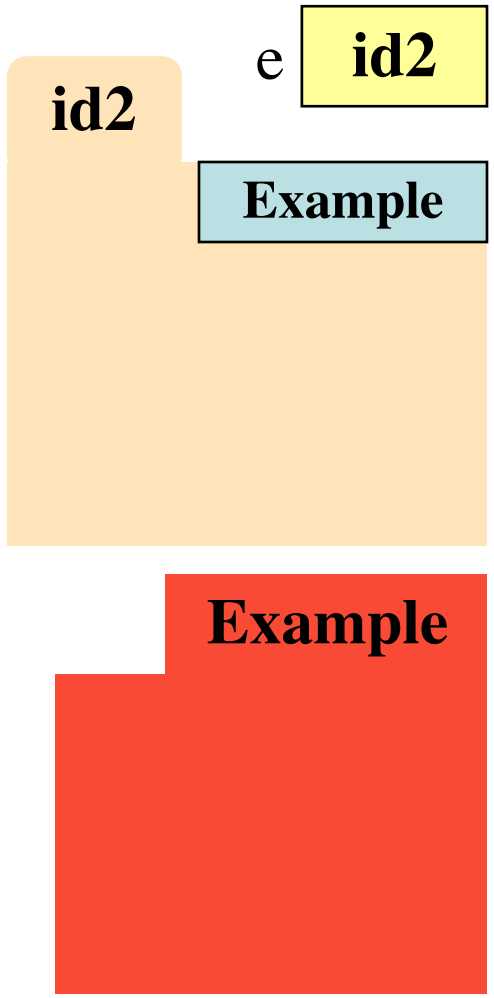
Example

Python creates after reading the class definition

Recall: Constructors

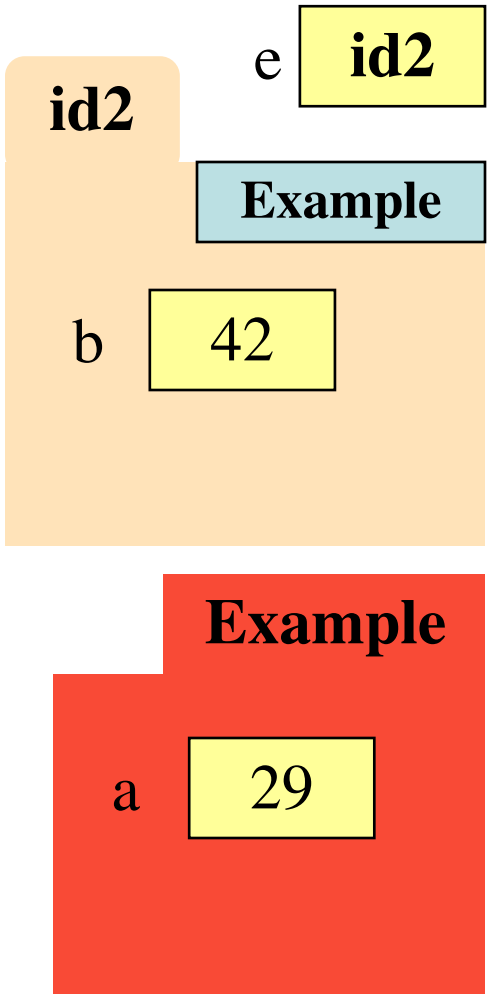
- Function to create new instances
  - Function name == class name
  - Created for you automatically
- Calling the constructor:
  - Makes a new object folder
  - Initializes attributes
  - Returns the id of the folder
- By default, takes no arguments
  - `e = Example()`

Will come back to this



Instances and Attributes

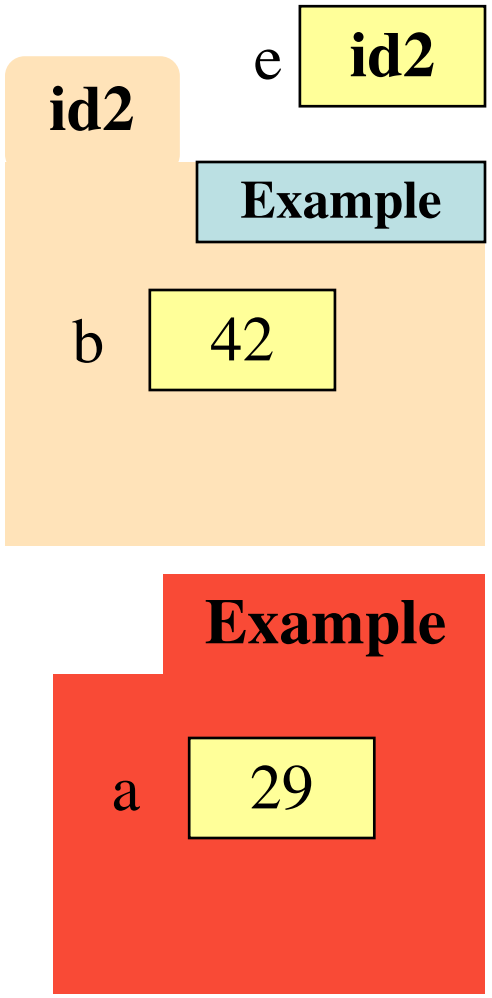
- Assignments add object attributes
  - <object>.<att> = <expression>
  - **Example**: `e.b = 42`
- Assignments can add class attributes
  - <class>.<att> = <expression>
  - **Example**: `Example.a = 29`
- Objects can access class attributes
  - **Example**: `print(e.a)`
  - But assigning it creates object attribute
  - **Example**: `e.a = 10`
- **Rule**: check object first, then class



Instances and Attributes

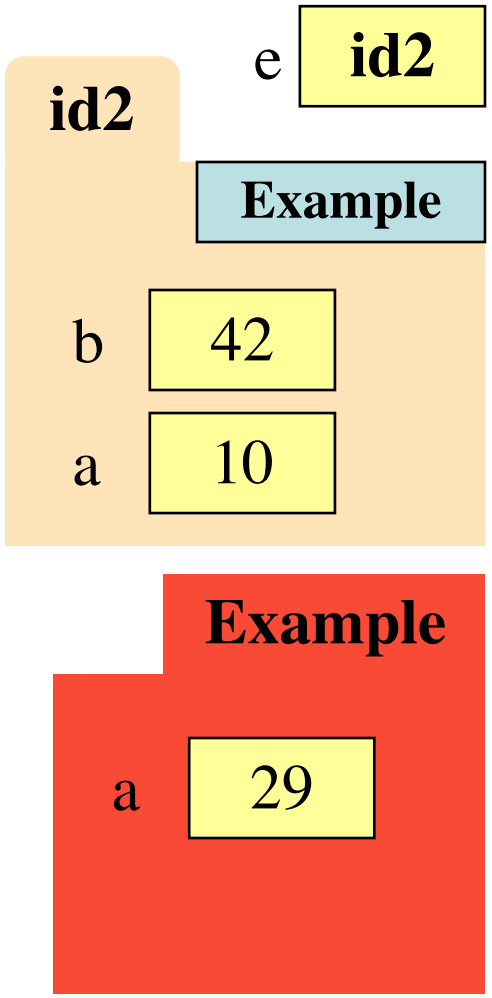
- Assignments add object attributes
  - <object>.<att> = <expression>
  - **Example**: `e.b = 42`
- Assignments can add class attributes
  - <class>.<att> = <expression>
  - **Example**: `Example.a = 29`
- Objects can access class attributes
  - **Example**: `print(e.a)`
  - But assigning it creates object attribute
  - **Example**: `e.a = 10`
- **Rule**: check object first, then class

Not how usually done



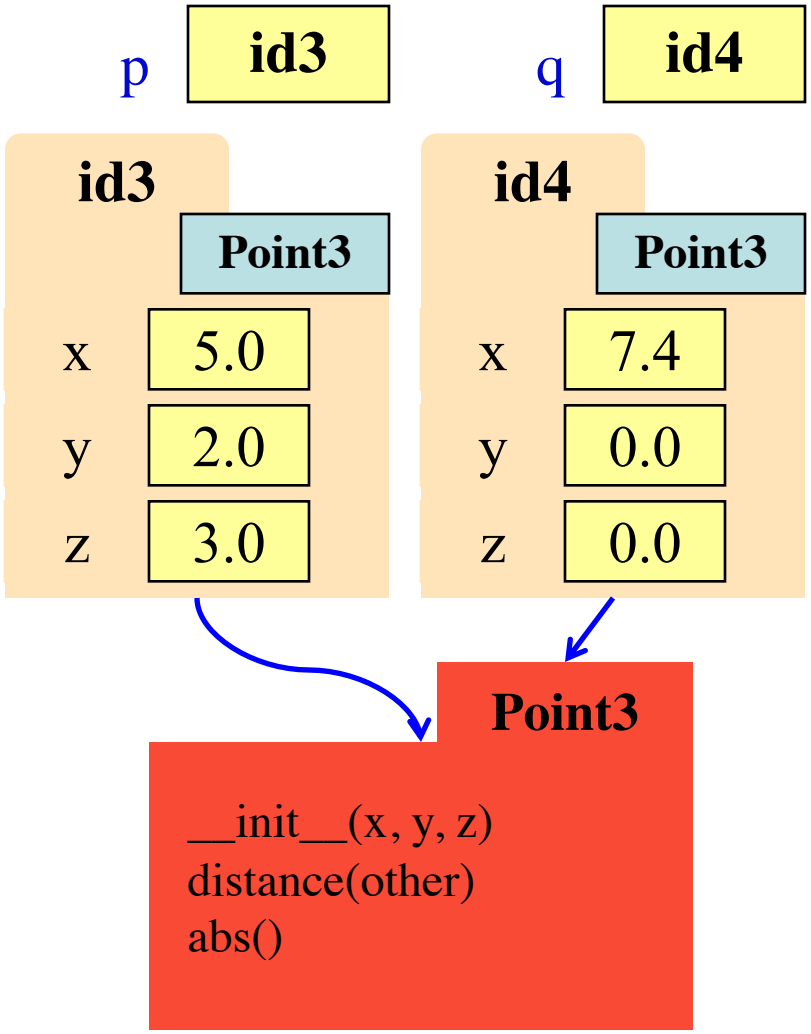
Instances and Attributes

- Assignments add object attributes
  - <object>.<att> = <expression>
  - **Example**: `e.b = 42`
- Assignments can add class attributes
  - <class>.<att> = <expression>
  - **Example**: `Example.a = 29`
- Objects can access class attributes
  - **Example**: `print(e.a)`
  - But assigning it creates object attribute
  - **Example**: `e.a = 10`
- **Rule**: check object first, then class



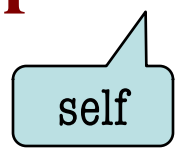
Recall: Objects can have Methods

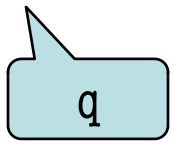
- **Method**: function tied to object
  - Function call: <function-name>(<arguments>)
  - Method call: <object-variable>.<function-call>
- **Example**: `p.distance(q)`
  - Both `p` and `q` act as arguments
  - Very much like `distance(p, q)`
- For most Python objects
  - **Attributes** are in **object** folder
  - **Methods** are in **class** folder



Method Definitions

- Looks like a function def
  - But indented *inside* class
  - The first parameter is always called self
- In a method call:
  - Parentheses have one less argument than parameters
  - The object in front is passed to parameter self
- **Example**: a.distance(b)

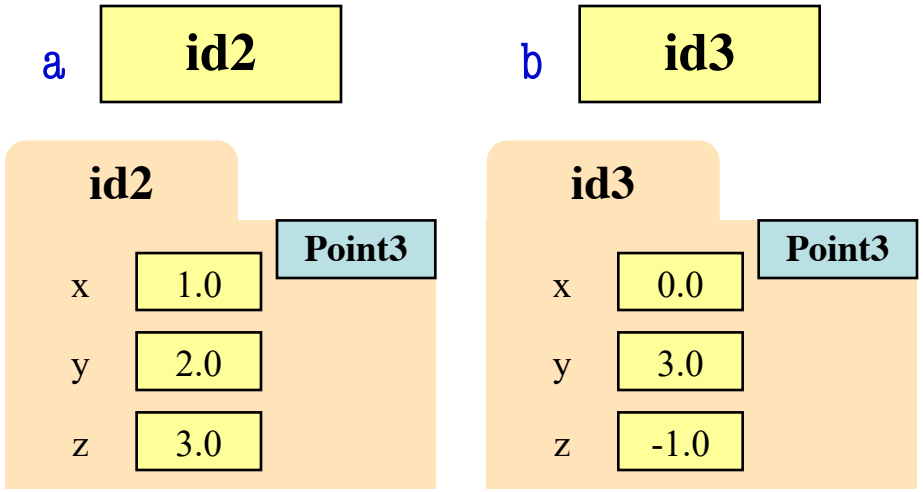





```
class Point3(object):
    """
    Instances are points in 3d space
    """
    def distance(self,q):
        """Returns: dist from self to q
        Precondition: q a Point3"""
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        return math.sqrt(sqrdst)
```

Methods Calls

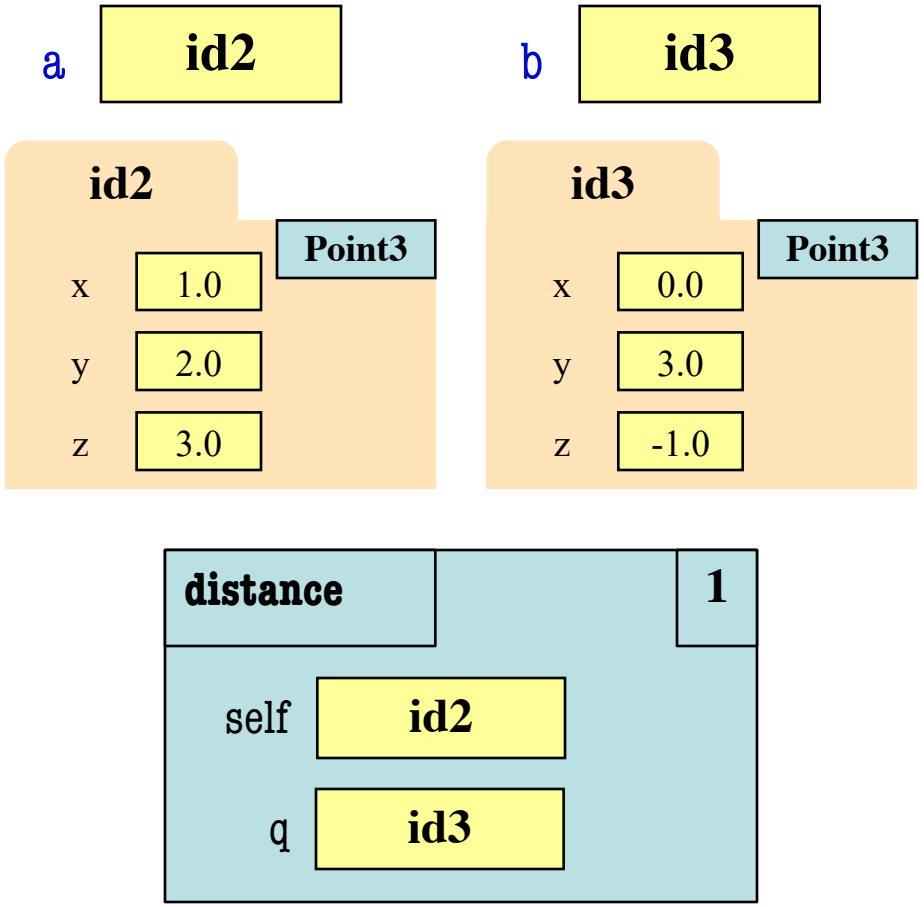
- **Example**: a.distance(b)




```
class Point3(object):
    """
    Instances are points in 3d space
    """
    def distance(self,q):
        """Returns: dist from self to q
        Precondition: q a Point3"""
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        return math.sqrt(sqrdst)
```

Methods Calls

- **Example**: a.distance(b)




```
class Point3(object):
    """
    Instances are points in 3d space
    """
    def distance(self,q):
        """Returns: dist from self to q
        Precondition: q a Point3"""
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        return math.sqrt(sqrdst)
```