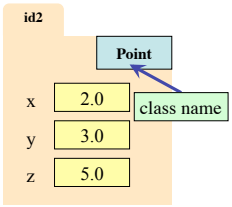
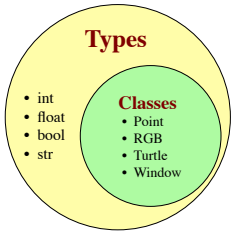
Recall: Classes are Types for Objects

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

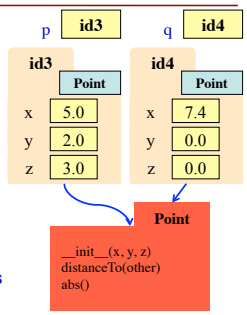


- Classes are how we add new types to Python



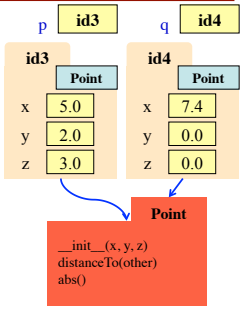
Recall: Objects can have Methods

- **Method**: function tied to object
  - Function call: <function-name>(<arguments>)
  - Method call: <object-variable>.<function-call>
  - Use of a method is a *method call*
- **Example**: p.distanceTo(q)
  - Both p and q act as arguments
  - Very much like distanceTo(p, q)
- Methods (often) in **class folders**

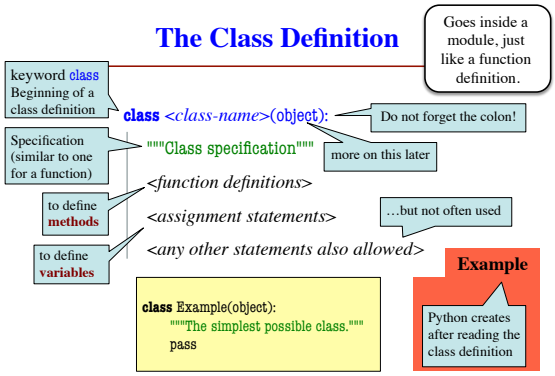


Name Resolution for Objects

- ⟨*object*⟩.⟨*name*⟩ means
  - Go the folder for *object*
  - Find attribute/method *name*
  - If missing, check **class folder**
  - If not in either, raise error
- For most Python objects
  - **Attributes** are in **object** folder
  - **Methods** are in **class** folder
- Rules can be broken…
  (but not in this class)

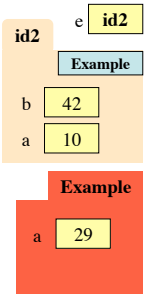


The Class Definition



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



Invariants

- Properties of an attribute that must be true
- Works like a precondition:
  - If invariant satisfied, object works properly
  - If not satisfied, object is "corrupted"
- **Examples**:
  - Point class: all attributes must be floats
  - RGB class: all attributes must be ints in 0..255
- Purpose of the **class specification**

The Class Specification

```
class Worker(object):
    """An instance is a worker in an organization.

    Instance has basic worker info, but no salary information.

    ATTRIBUTES:
    lname:  Worker's last name.  [str]
    ssn:    Social security no.  [int in 0..999999999]
    boss:   Worker's boss.       [Worker, or None if no boss]
```

Short summary

More detail

Attribute list

Description

Invariant

Attribute Name

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.distanceTo(b)

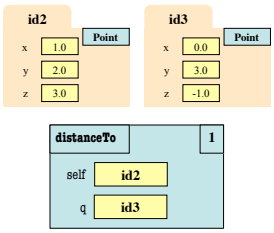
self q

```
class Point(object):
    """Instances are points in 3d space
        x: x coord [float]
        y: y coord [float]
        z: z coord [float]    """
    def distanceTo(self,q):
        """Returns: dist from self to q
        Precondition: q a Point"""
        assert type(q) == Point
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        return math.sqrt(sqrdst)
```

Methods Calls

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

a id2 b id3



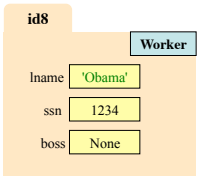

```
class Point(object):
    """Instances are points in 3d space
        x: x coord [float]
        y: y coord [float]
        z: z coord [float]    """
    def distanceTo(self,q):
        """Returns: dist from self to q
        Precondition: q a Point"""
        assert type(q) == Point
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        return math.sqrt(sqrdst)
```

Special Method: __init__

```
w = Worker('Obama', 1234, None)
```

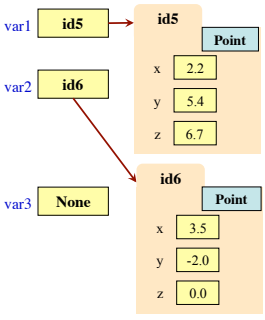
```
def __init__(self, n, s, b):
    """Initializer: creates a Worker

    Has last name n, SSN s, and boss b

    Precondition: n a string, s an int in
    range 0..999999999, and b either
    a Worker or None.
    self.lname = n
    self.ssn  = s
    self.boss = b
```

Called by the constructor



Aside: The Value None

- The boss field is a problem.
  - boss refers to a Worker object
  - Some workers have no boss
  - Or maybe not assigned yet (the buck stops there)
- **Solution**: use value None
  - **None**: Lack of (folder) name
  - Will reassign the field later!
- Be careful with None values
  - var3.x gives error!
  - There is no name in var3
  - Which Point to use?



Making Arguments Optional

- We can assign default values to __init__ arguments
  - Write as assignments to parameters in definition
  - Parameters with default values are optional
- **Examples**:
  - p = Point() # (0,0,0)
  - p = Point(1,2,3) # (1,2,3)
  - p = Point(1,2) # (1,2,0)
  - p = Point(y=3) # (0,3,0)
  - p = Point(1,z=2) # (1,0,2)

```
class Point(object):
    """Instances are points in 3d space
        x: x coord [float]
        y: y coord [float]
        z: z coord [float]    """
    def __init__(self,x=0,y=0,z=0):
        """Initializer: makes a new Point
        Precondition: x,y,z are numbers"""
        self.x = x
        self.y = y
        self.z = z
    ...
```