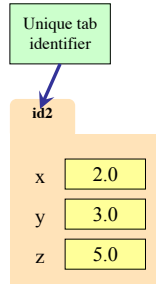


Recall: Objects as Data in Folders

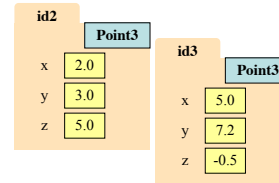
- An object is like a **manila folder**
- It contains other variables
 - Variables are called **attributes**
 - Can change values of an attribute (with assignment statements)
- It has a “tab” that identifies it
 - Unique number assigned by Python
 - Fixed for lifetime of the object



Classes Have Folders Too

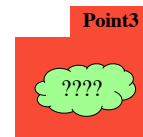
Object Folders

- Separate for each *instance*



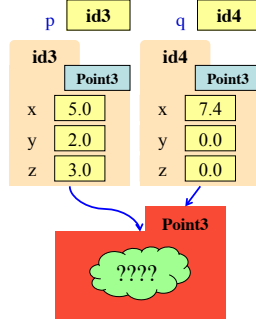
Class Folders

- Data common to all instances

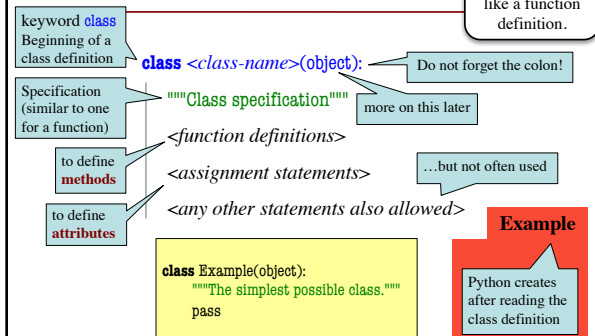


Name Resolution for Objects

- `<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**

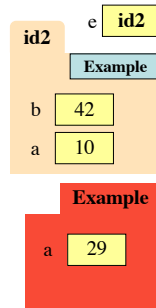


The Class Definition

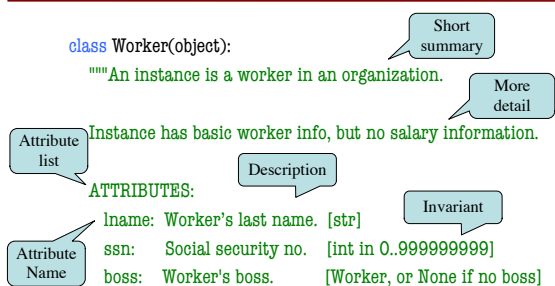


Instances and Attributes

- 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



The Class Specification



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)

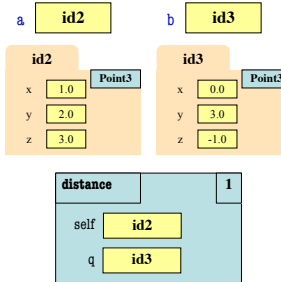
self

q

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

Special Method: `__init__`

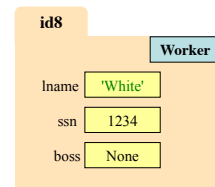
two underscores
w = WORKER, WILL... 1234 None

don't forget self

```
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
```

use self to assign attributes

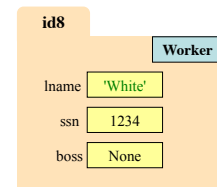
Called by the constructor



Evaluating a Constructor Expression

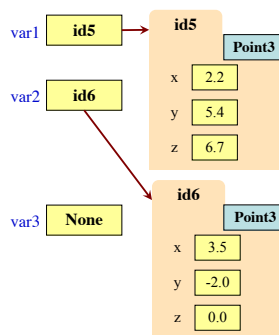
Worker('White', 1234, None)

- Creates a new object (folder) of the class Worker
 - Instance is initially empty
- Puts the folder into heap space
- Executes the method `__init__`
 - Passes folder name to self
 - Passes other arguments in order
 - Executes the (assignment) commands in initializer body
- Returns the object (folder) name



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 = Point3()` # (0,0,0)
- `p = Point3(1,2,3)` # (1,2,3)
- `p = Point3(1,2)` # (1,2,0)
- `p = Point3(y=3)` # (0,3,0)
- `p = Point3(1,z=2)` # (1,0,2)

```
class Point3(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
    ...
```