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

```python
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]
```

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
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)`

```python
def a.distance(b):
    """Initializer: creates a Worker
    Precondition: q a Point3"
    sqrdst = (self.x-q.x)**2 + (self.y-q.y)**2 + (self.z-q.z)**2
    return math.sqrt(sqrdst)
```

Special Method: `__init__`

```python
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

```python
id8 = Worker(n='White', ssn=1234, boss=None)
```

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?

```python
id5 = Point3(x=2.2, y=5.4, z=6.7)
id6 = None
id7 = Point3(x=3.5, y=-2.0, z=0.0)
```

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

```python
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, w=0):
    """Initializer: makes a new Point
    Precondition: x,y,z are numbers"
    self.x = x
    self.y = y
    self.z = z
    """
```