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

![Unique tab identifier](image)

Classes Have Folders Too

**Object Folders**
- Separate for each instance

**Class Folders**
- Data common to all instances

![Object Folders](image)  
![Class Folders](image)

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**

![What is in the class folder?](image)

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](image)

The Class Definition

- **class** `<class-name>(object)`
  - **Keyword class**: Beginning of a class definition
  - **Specification** (similar to one for a function)
  - **<function definitions>**
  - **<assignment statements>**
  - **<any other statements also allowed>**

![The Class Definition](image)

The Class Specification

- **class** `Worker(object)`
  - **Short summary**
    - **Invariant**: Worker has basic info, but no salary.
  - **More detail**
    - **ATTRIBUTES**
      - `name`: Worker’s last name. [str]
      - `ssn`: Social security no. [int in 0..999999999]
      - `boss`: Worker’s boss. [Worker, or None if no boss]
**Method Definitions**

- Looks like a function `def` but indented `inside`
  - 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)`

```
def a.distance(b):
    sqrdst = ((self.x-b.x)**2 + (self.y-b.y)**2 + (self.z-b.z)**2)
    return math.sqrt(sqrdst)
```

**Methods Calls**

```
Example: a.distance(b)
```

**Special Method: `__init__`**

- Called by the constructor

```
def __init__(self, n, s, b):
    """Initializer: creates a Worker"
    self.lname = n
    self.ssn = s
    self.boss = b
```

**Evaluating a Constructor Expression**

```
Worker('White', 1234, None)
```

1. Creates a new object (folder) of the class `Worker`
   - Instance is initially empty
2. Puts the folder into heap space
3. Executes the method `__init__`
   - Passes folder name to `self`
   - Passes other arguments in order
4. 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 Pointer to use?

```
Example

\[ (0,0,0) \quad (1,2,3) \quad (1,2,0) \quad (0,3,0) \quad (1,0,2) \]
```

**Making Arguments Optional**

- We can assign default values to `__init__` arguments
  - Write as assignments to parameters in definition
  - Parameters with default values are optional

```
def __init__(self, x=0, y=0, z=0):
    """Initializer: makes a new Point"
    self.x = x
    self.y = y
    self.z = z
```

```python
class Point3(object):
    """Instances are points in 3d space"
    x: x coord (float)
    y: y coord (float)
    z: z coord (float)
```

```python
p = Point3(x=1, y=2, z=3)
p = Point3(1, 2, 3)
p = Point3((1, 2, 3))
p = Point3([1, 2, 3])
p = Point3(1, 2, 3)
```

```
class Point3(object):
    """Instances are points in 3d space"
    x: x coord (float)
    y: y coord (float)
    z: z coord (float)
```

```python
p = Point3(x=1, y=2, z=3)
p = Point3(1, 2, 3)
p = Point3((1, 2, 3))
p = Point3([1, 2, 3])
p = Point3(1, 2, 3)
```