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

id2

x	2.0
y	3.0
z	5.0

1

Classes Have Folders Too

Object Folders	Class Folders
• Separate for each *instance*	• Data common to all instances

id2 — Point3
x	2.0
y	3.0
z	5.0

id3 — Point3
x	5.0
y	7.2
z	-0.5

Point3
????

2

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*

p id3   q id4

id3 — Point3
x	5.0
y	2.0
z	3.0

id4 — Point3
x	7.4
y	0.0
z	0.0

Point3
????

3

The Class Definition

Goes inside a module, just like a function definition.

keyword class
Beginning of a class definition

Specification (similar to one for a function)

to define **methods**

to define **attributes**

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

Do not forget the colon!

"""Class specification"""

more on this later

<function definitions>

<assignment statements>

…but not often used

<any other statements also allowed>

Example

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

Python creates after reading the class definition

4

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

e id2

id2 — Example
| b | 42 |
| a | 10 |

Example
| a | 29 |

5

The Class Specification

```
class Worker(object):
    """A class representing a worker in a certain organization

    Instance has basic worker info, but no salary information.

    Attribute lname: The worker last name
    Invariant: lname is a string

    Attribute ssn: The Social Security number
    Invariant: ssn is an int in the range 0..999999999

    Attribute boss: The worker's boss
    Invariant: boss is an instace of Worker, or None if no boss"""
```

Short summary

More detail

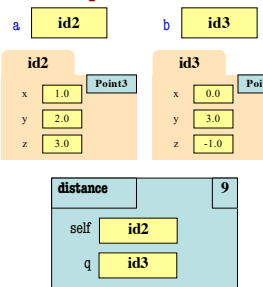Description

Invariant

6

Method Definitions

- Looks like a function `def`
  - Indented *inside* class
  - First param is always `self`
  - But otherwise the same
- In a **method call**:
  - One less argument in ()
  - Obj in front goes to `self`
- **Example**: `a.distance(b)`
  - `self`
  - `q`

```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float    """
6.      def distance(self,q):
7.          """Returns dist from self to q
8.          Precondition: q a Point3"""
9.          assert type(q) == Point3
10.         sqrdst = ((self.x-q.x)**2 +
11.                   (self.y-q.y)**2 +
12.                   (self.z-q.z)**2)
13.         return math.sqrt(sqrdst)
```

7

Methods Calls

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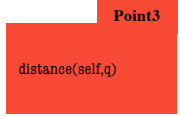


```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float    """
6.      def distance(self,q):
7.          """Returns dist from self to q
8.          Precondition: q a Point3"""
9.          assert type(q) == Point3
10.         sqrdst = ((self.x-q.x)**2 +
11.                   (self.y-q.y)**2 +
12.                   (self.z-q.z)**2)
13.         return math.sqrt(sqrdst)
```

8

Methods and Folders

- Function definitions…
  - make a folder in heap
  - assign name as variable
  - variable in global space
- Methods are similar...
  - Variable in **class folder**
  - But otherwise the same
- **Rule of this course**
  - Put header in class folder
  - Nothing else!
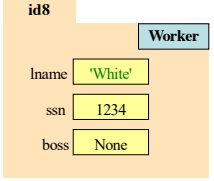
```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float    """
6.      def distance(self,q):
        ....
```
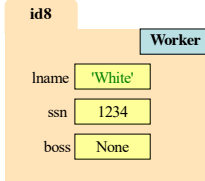
Point3

distance(self,q)

9

Special Method: __init__

two underscores

w = Worker(... 1234, None)

don't forget self

Called by the constructor

```
def __init__(self, n, s, b):
    """Initializes a Worker object

    Has last name n, SSN s, and boss b

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

use `self` to assign attributes

id8 Worker

lname	'White'
ssn	1234
boss	None

10

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
   - Executes the (assignment) commands in initializer body
4. Returns the object (folder) name

id8 Worker

lname	'White'
ssn	1234
boss	None

11

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)

```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float    """
6.
7.      def __init__(self,x=0,y=0,z=0):
8.          """Initializes a new Point3
9.          Precond: x,y,z are numbers"""
10.         self.x = x
11.         self.y = y
12.         self.z = z
13.         ...
```

12