

Lecture 17

Classes

Announcements for This Lecture

Assignments

- **A4 Thursday** at midnight
 - Hopefully you are on Task 4
 - Minor extension for reasons
- Will post **A5** on Thursday
 - Written assignment like A2
 - Needs material from Tues
- Will post **A6** on Nov 3.
 - Not due until November 20
 - Want to avoid exam crunch

Lab this Week

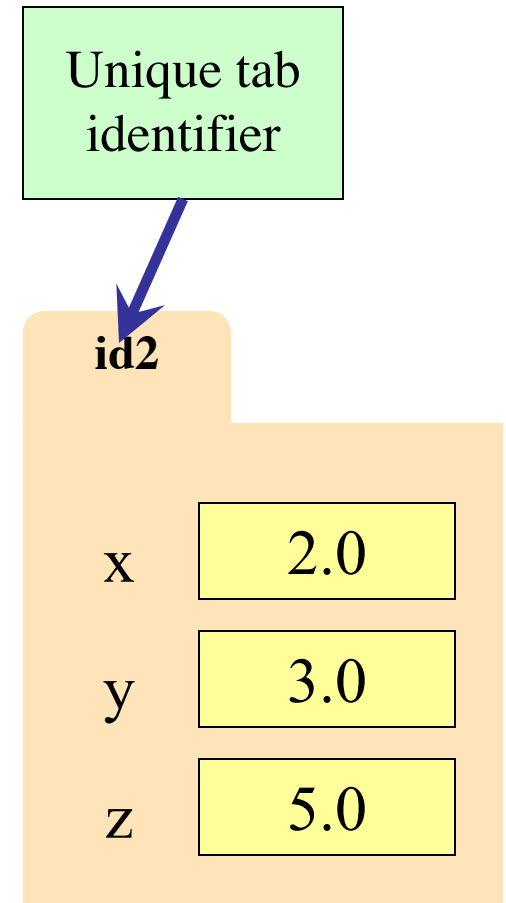
- More prelim exercises
 - This time for-loops
 - Also tables, dictionaries

Exams

- Last week for regrades
 - Limit them to valid issues
- Getting closer to prelim 2

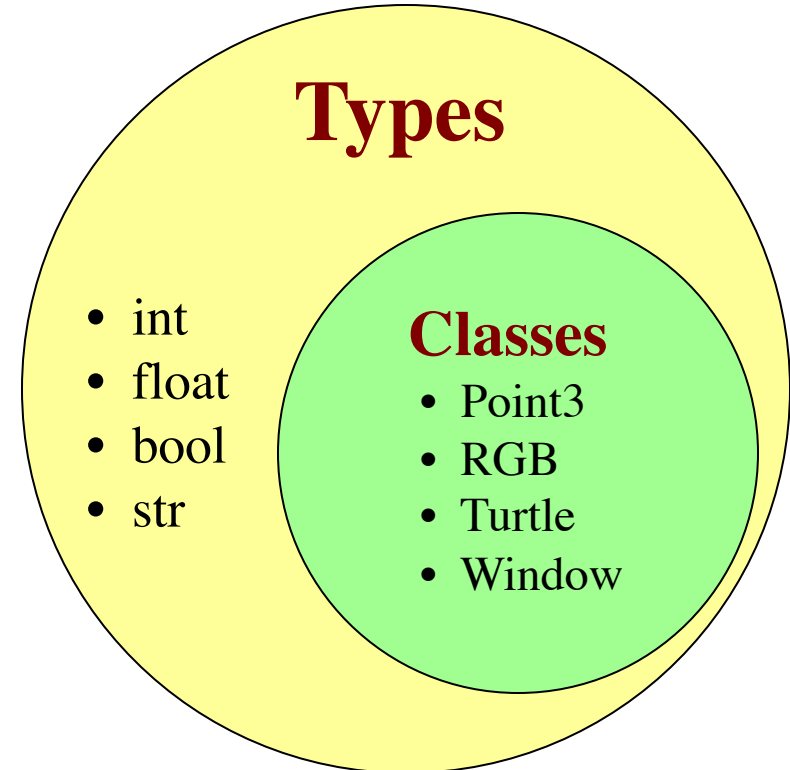
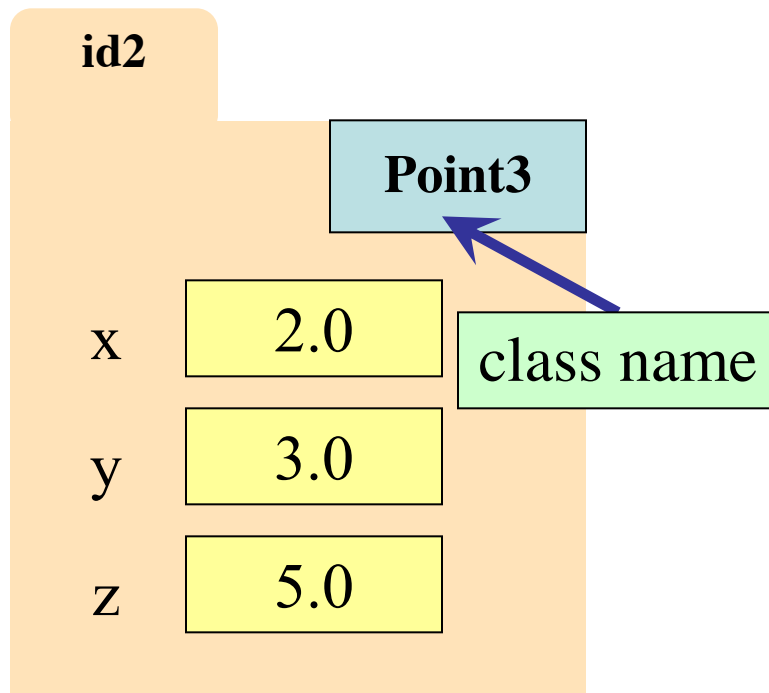
Recall: Objects as Data in Folders

- 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



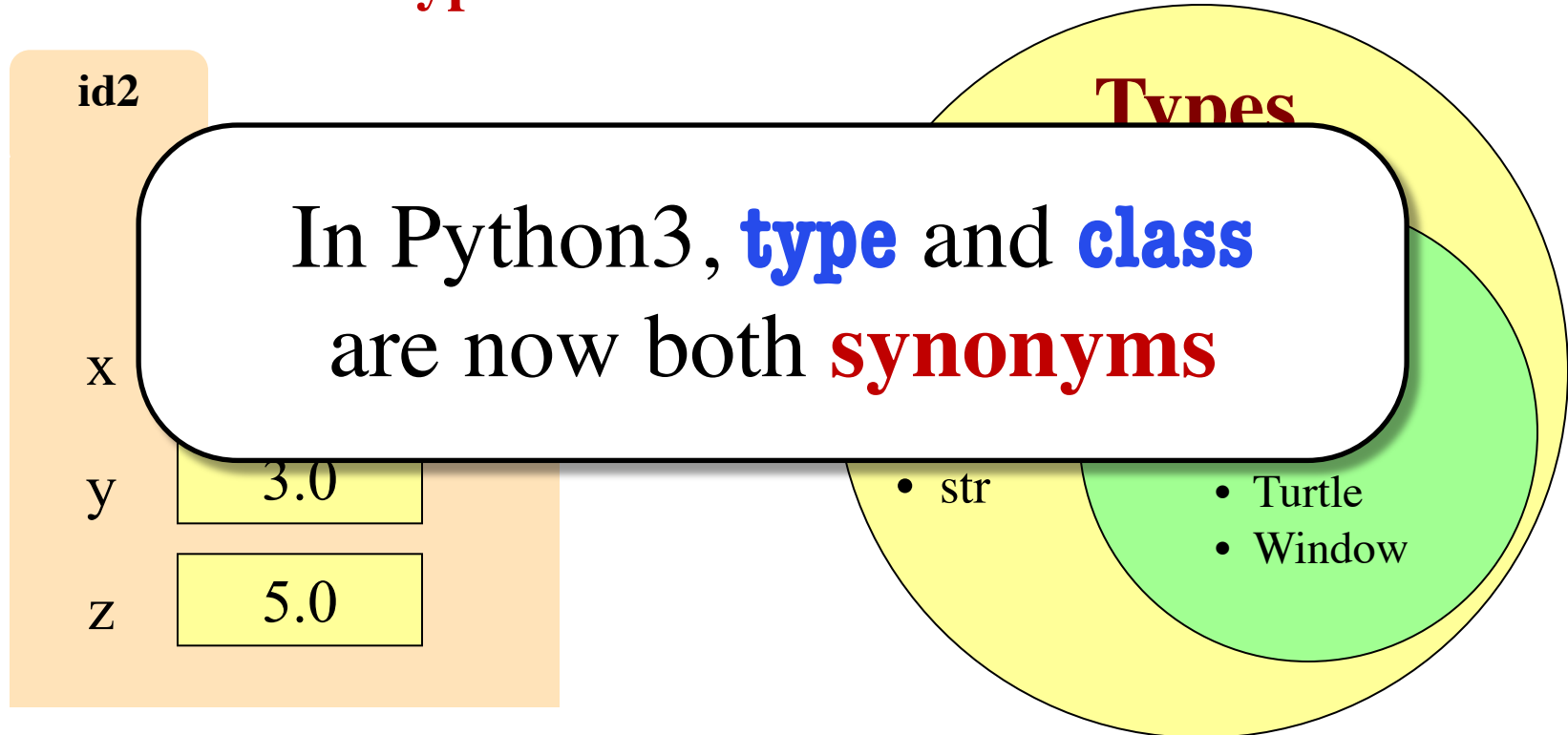
Recall: Classes are Types for Objects

- Values must have a type
 - An **object** is a **value**
 - A **class** is its **type**
- Classes are how we add new types to Python



Recall: Classes are Types for Objects

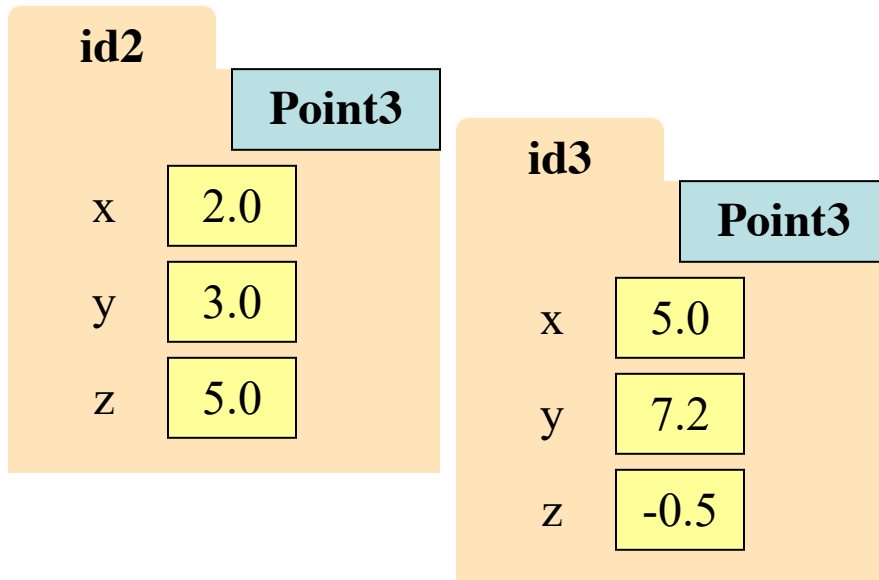
- Values must have a type
 - An **object** is a **value**
 - A **class** is its **type**
- Classes are how we add new types to Python



Classes Have Folders Too

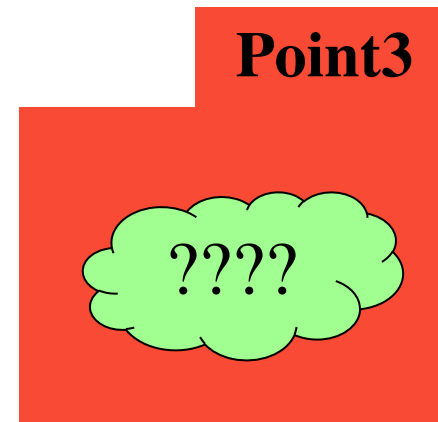
Object Folders

- Separate for each *instance*



Class Folders

- Data common to all instances



The Class Definition

Goes inside a module, just like a function definition.

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

"""Class specification"""

<function definitions>

<assignment statements>

<any other statements also allowed>

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

Example

The Class Definition

Goes inside a module, just like a function definition.

keyword **class**
Beginning of a class definition

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

Do not forget the colon!

Specification
(similar to one for a function)

"""Class specification"""

more on this later

<function definitions>

to define **methods**

<assignment statements>

...but not often used

to define **attributes**

<any other statements also allowed>

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

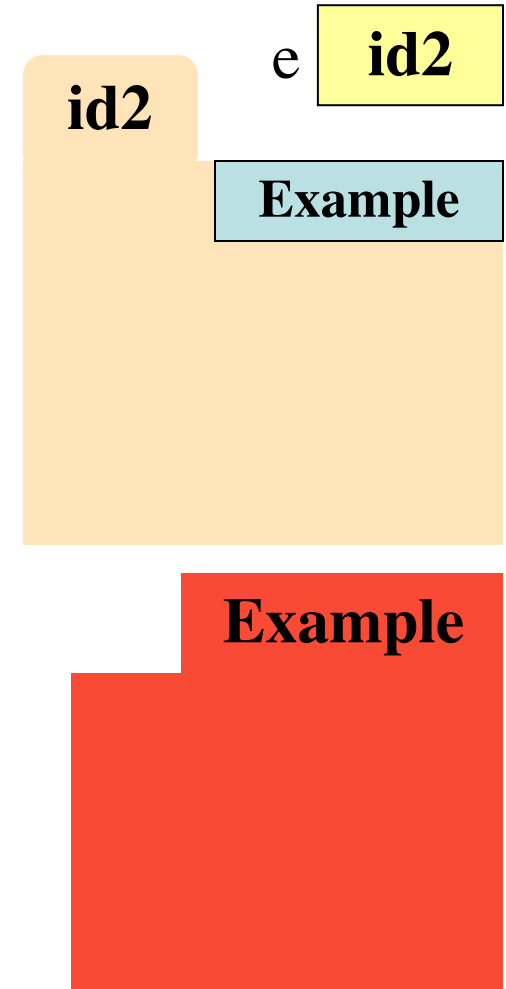
Example

Python creates after reading the class definition

Recall: Constructors

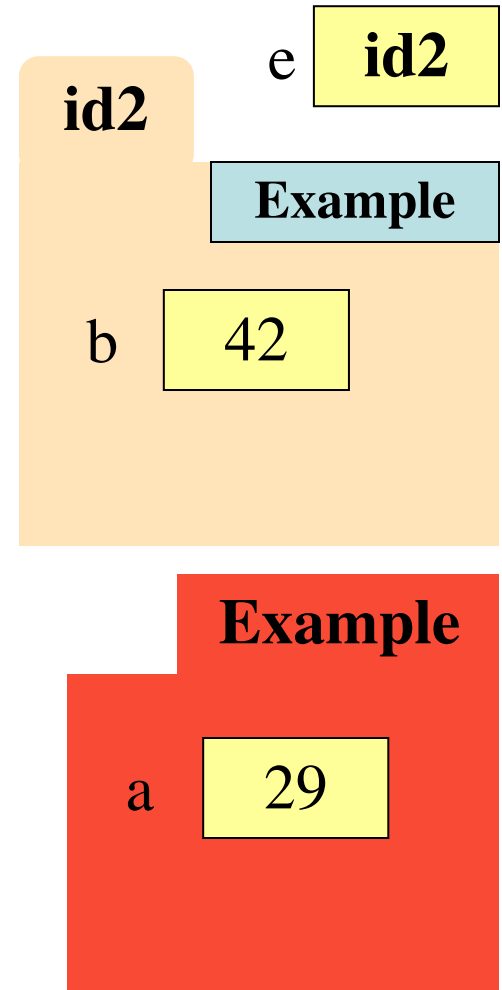
- Function to create new instances
 - Function name == class name
 - Created for you automatically
- Calling the constructor:
 - Makes a new object folder
 - Initializes attributes
 - Returns the id of the folder
- By default, takes no arguments
 - `e = Example()`

Will come back to this

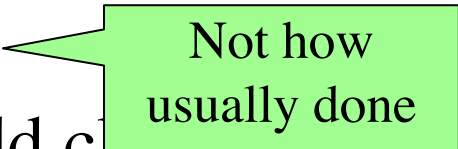


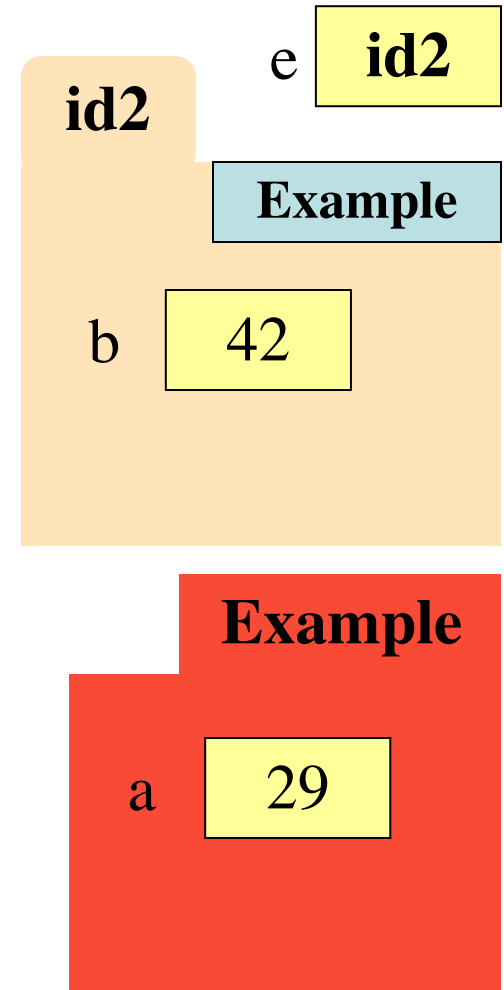
Instances and Attributes

- 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



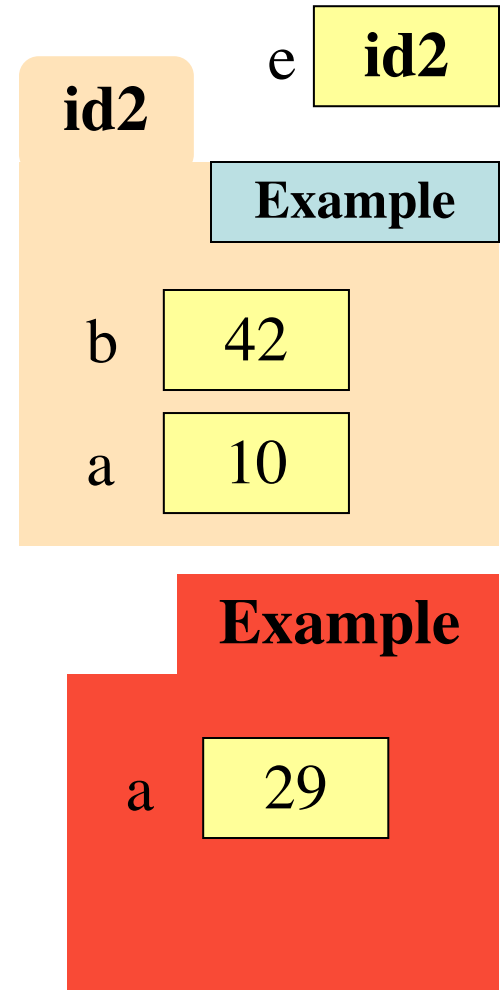
Instances and Attributes

- 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

- 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:**
 - **Point3** 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):
```

```
    """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"""
```

The Class Specification

```
class Worker(object):
```

```
    """A class representing a worker in a certain organization
```

Short
summary

```
    Instance has basic worker info, but no salary information.
```

More
detail

```
    Attribute lname: The worker last name
```

Description

```
    Invariant: lname is a string
```

Invariant

```
    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"""
```

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
```

```
    Invariant
```

**Warning: New format this year.
Old exams will be very different.**

```
    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"""
```


Recall: Objects can have Methods

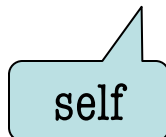
- Object before the name is an *implicit* argument

- **Example:** distance

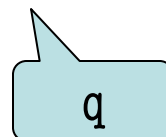
```
>>> p = Point3(0,0,0)    # First point
>>> q = Point3(1,0,0)    # Second point
>>> r = Point3(0,0,1)    # Third point
>>> p.distance(r)        # Distance between p, r
1.0
>>> q.distance(r)        # Distance between q, r
1.4142135623730951
```

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

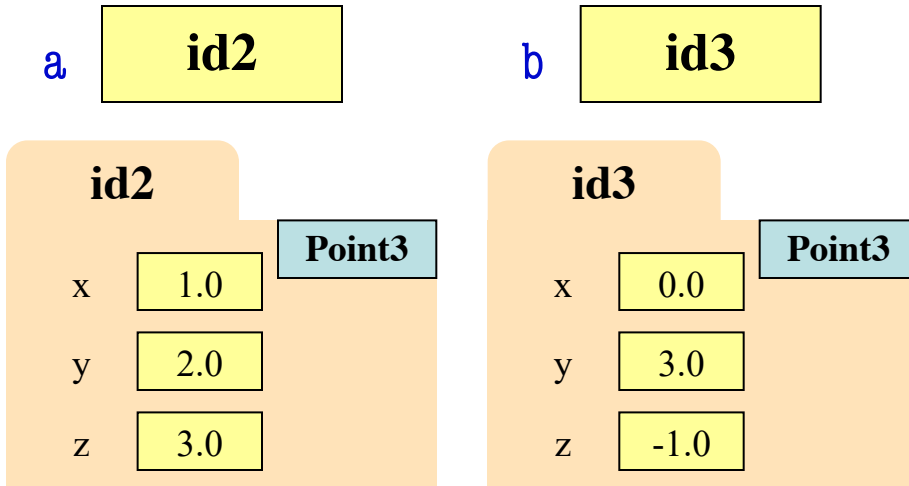


q

```
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)
```

Methods Calls

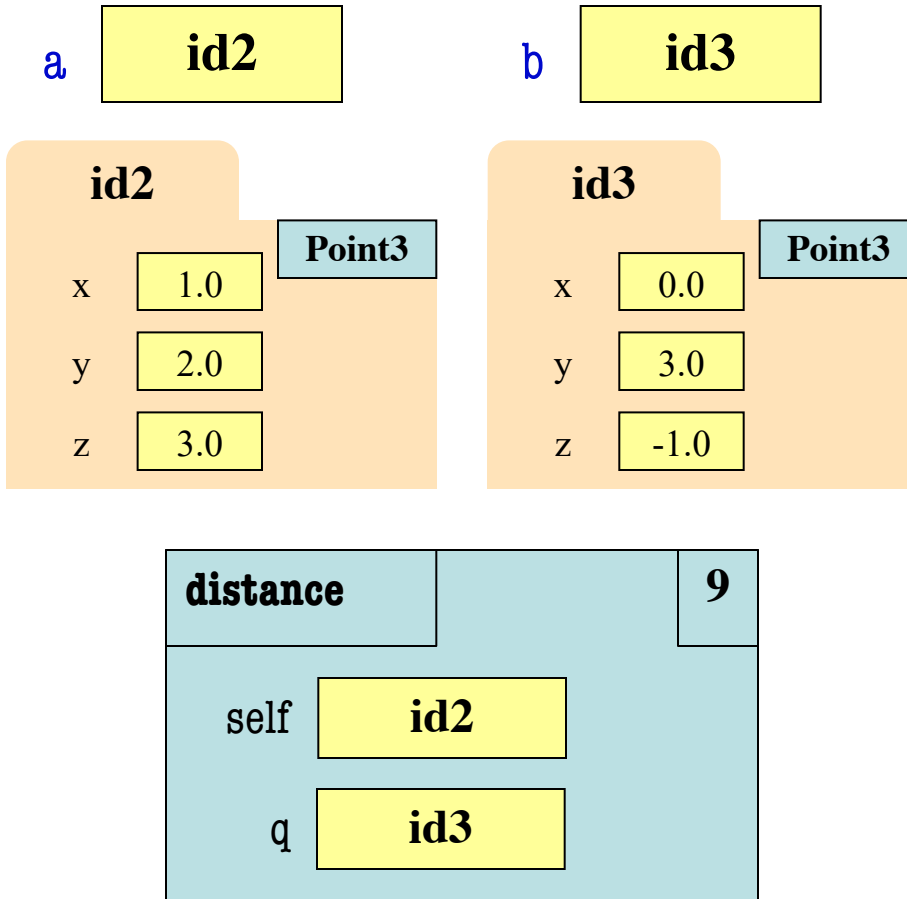
- **Example:** `a.distance(b)`



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)`

Methods Calls

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

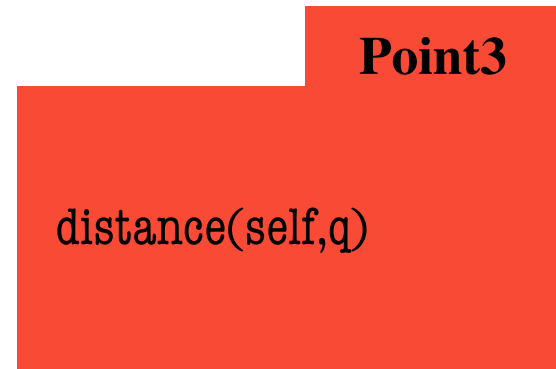


1. `class Point3(object):`
2. `"""Class for points in 3d space`
3. `Invariant: x is a float`
4. `Invariant y is a float`
5. `Invariant z is a float """`
6. `def distance(self,q):`
7. `"""Returns dist from self to q`
8. `Precondition: q a Point3"""`
9. `assert type(q) == Point3`
10. `sqrdst = ((self.x-q.x)**2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

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):
    ....
```



Methods and Folders

Visualize

Execute Code

Edit Code

Heap primitives Use arrows

```
→ 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)
```

Globals

global
Point3 | id1

Frames

Objects

id1:Point3 class
[hide attributes](#)
distance | id2:function
distance(self, q)

Just this

<< First < Back Program terminated Forward > Last >>

→ line that has just executed

→ next line to execute

Initializing the Attributes of an Object (Folder)

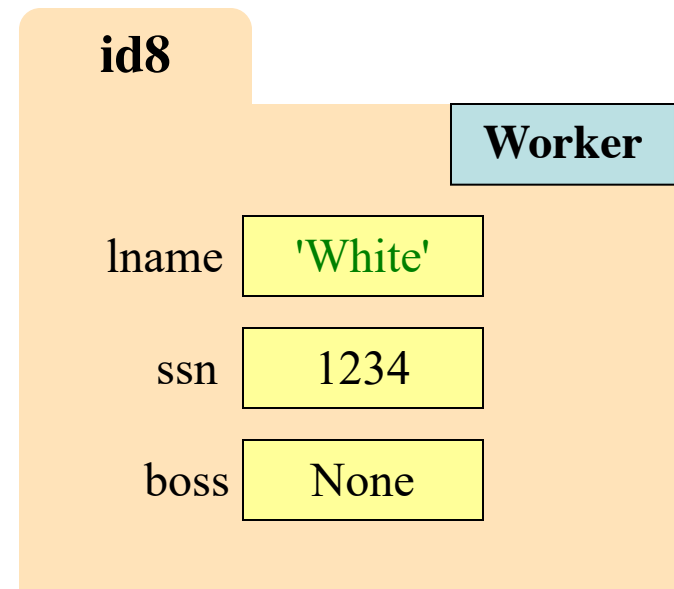
- Creating a new Worker is a multi-step process:
 - `w = Worker()` ← Instance is empty
 - `w.lname = 'White'`
 - ...
- Want to use something like
 - `w = Worker('White', 1234, None)`
 - Create a new Worker **and** assign attributes
 - lname to 'White', ssn to 1234, and boss to None
- Need a **custom constructor**

Special Method: `__init__`

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

```
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
```

Called by the constructor



Special Method: `__init__`

two underscores

```
w = Worker(White, 1234, None)
```

don't forget self

```
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

Called by the constructor

id8

Worker

lname 'White'

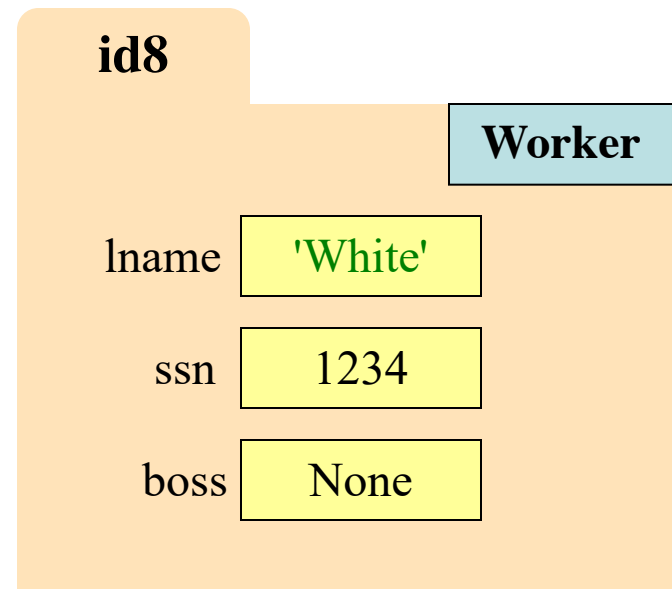
ssn 1234

boss None

Evaluating a Constructor Expression

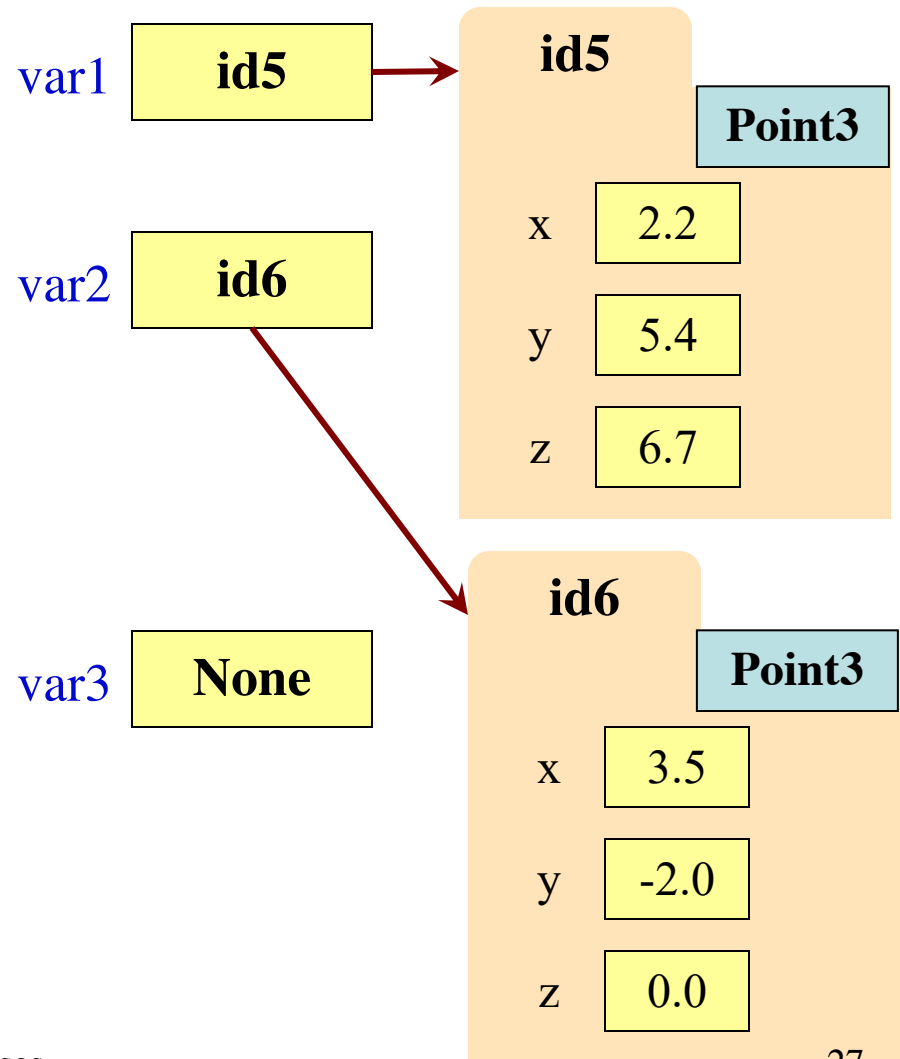
```
Worker('White', 1234, None)
```

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



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 Point3 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)

```
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.    ...
```

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)` Assigns in order
- `p = Point3(1,2)`
- `p = Point3(y=3)` Use parameter name when out of order
- `p = Point3(1,z=2)` Can mix two approaches

```
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
...

```

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()` Assigns in order
- `p = Point3(1,2)`
- `p = Point3(y=3)` Use parameter name when out of order
- `p = Point3(1,z=2)` Can mix two approaches

```
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 point
9.         p
10.        """
11.         self.x = x
12.         self.y = y
13.         self.z = z
...

```

Not limited to methods.
Can do with any function.