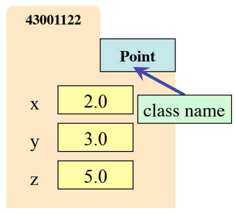
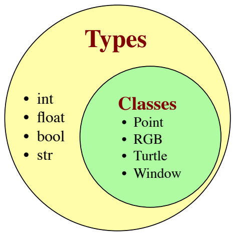
Recall: Classes are Types for Objects

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





The Class Definition

- Defines the format of any object of that class
- Everything is indented under the class name

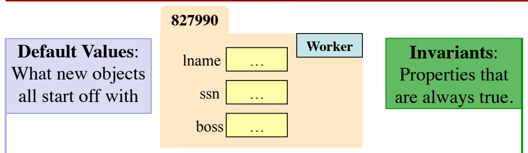
```
class <class-name>(object):
    """Class specification"""
    definitions of fields
    definitions of methods
    (in any order)
```

- Simplest class definition:

```
class Example(object):
    """Does nothing"""
    pass
```

This definition goes inside of a module, just like a function definition does.

Fields: Adding Attributes to a Class



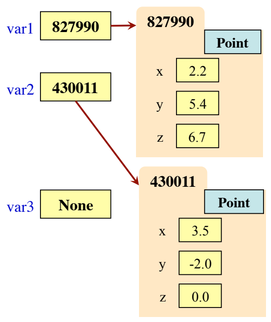
Default Values: What new objects all start off with

Invariants: Properties that are always true.

```
class Worker(object):
    """An instance is a worker in a certain organization."""
    lname = ''      # Last name (string, '' if unknown)
    ssn = 0         # Social security # (int in range 0..999999999)
    boss = None     # Immediate boss (Worker object; None if none)
```

The Value None

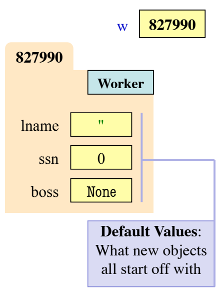
- The boss field is a problem.
  - boss is a Worker object
  - But we are defining what the Worker class looks like
  - Cannot put a value in boss until the definition is done
- **Solution**: use value None
  - **None**: Lack of (folder) name
  - Will reassign the field later!
- Be careful with None variables
  - var3.x gives error!
  - There is no name in var3
  - Which Point to use?



Constructors

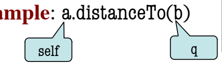
- Class definition creates a special (hidden) function
  - **Same name as the class**
  - (For now) no arguments
- Called the **constructor**
  - Makes new object of class
  - Returns the id of object
- Example:
  - Point()
  - Worker()

```
w = Worker()
```



Methods

- 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.distanceTo(b)

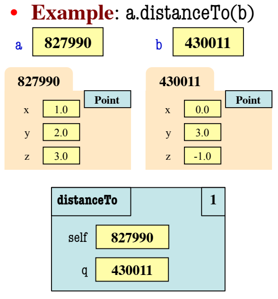


```
class Point(object):
    """Instances are points in 3d space"""
    x = 0.0 # x coord, float
    y = 0.0 # y coord, float
    z = 0.0 # z coord, float

    def distanceTo(self,q):
        """Returns: dist from self to q
        Precondition: q a Point"""
        assert type(q) == Point
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        return math.sqrt(sqrdst)
```

Methods Calls

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



```
class Point(object):
    """Instances are points in 3d space"""
    x = 0.0 # x coord, float
    y = 0.0 # y coord, float
    z = 0.0 # z coord, float
    def distanceTo(self,q):
        """Returns: dist from self to q
        Precondition: q a Point"""
        assert type(q) == Point
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        return math.sqrt(sqrdst)
```

Initializing the Fields of an Object (Folder)

- Creating a new Worker is a multi-step process:
  - w = new Worker() ← Fields are all defaults
  - w.lname = 'White'
  - ...
- Want to use something like

  w = Worker('White', 1234, None)
  - Create a new Worker **and** assign fields
  - lname to 'White', ssn to 1234, and boss to None
- Need a **custom constructor**

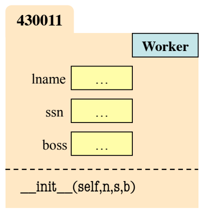
Special Method: __init__

```
def __init__(self, n, s, b):
    """Constructor: creates a Worker
    Instance 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
```

two underscores

don't forget self

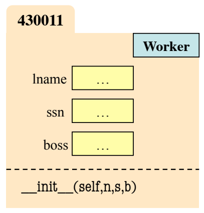
use self to access fields



How a Constructor Expression Works

Worker('White', 1, null)

1. Creates a new object (folder) of the class Worker
   - Fields set to default values
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 constructor body
4. Returns the object (folder) name as final value of expression



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

```
class Point(object):
    """Instances are points in 3d space"""
    x = 0.0 # x coord, float
    y = 0.0 # y coord, float
    z = 0.0 # z coord, float

    def __init__(self,x=0,y=0,z=0):
        """Constructor: makes a new Point
        Precondition: x,y,z are numbers"""
        self.x = x
        self.y = y
        self.z = z
    ...
```

What Does str() Do On Objects?

- Does **NOT** display contents
  ```
  >>> p = Point(1,2,3)
  >>> str(p)
  '<Point object at 0x1007a90>'
  ```
- To display contents, you must implement a special method
  - __str__ for str()
  - __repr__ for backquotes
  - If only implement __str__, backquotes do not work
  - If implement __repr__ but not __str__, str() uses it too

```
class Point(object):
    """Instances are points in 3d space"""
    ...
    def __str__(self):
        """Returns: string with contents"""
        return '('+self.x + ',' +
               self.y + ',' +
               self.z + ')'

    def __repr__(self):
        """Returns: unambiguous string"""
        return str(self.__class__)+
            str(self)
```