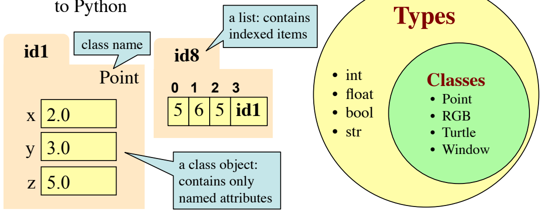
Recall: Classes are Types for Objects

- Objects must have types
  - Some types are built in (float, int, file, list, …)
  - Other types are defined by **classes**
  - Classes are how we add new types to Python



Machinery vs. use of machinery

- Classes in Python provide some very simple machinery, and very few constraints on how you use it.
- Learning to program with classes in Python means learning two things:
  1. how the machinery works (this lecture)
  2. some ways to use the machinery effectively (next lecture)

The Class Definition

Goes inside a module, just like a function definition.

keyword class indicates a class definition

docstring, just like a function definition

to define **methods**

to define **variables**

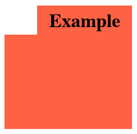
```
class <class-name>(object):
    """Class specification"""
    <function definitions>
    <assignment statements>
    <any other statements also allowed>
```

don't forget the colon!

more on this later

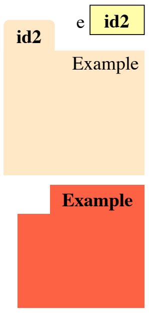
…but not often used

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



Instances and attributes

- You can create *instances* of the class:
  `e = Example()` a "constructor expression"
  - Creates a new, empty object
- and access *attributes* of the class:
  ```
  Example.a = 29
  print Example.a
  ```
  not the way we normally create class attributes! …more later
  - Writing to one creates a new attribute in the class
- and access *attributes* of an instance:
  ```
  e.b = 42
  print e.b
  ```
  not the way we normally create instance attributes! …more later
  - **Rule: look first in the instance, then the class**
  - Writing to one creates a new attribute in the instance
- and that's pretty much it!



Populating a class with methods

Everything defined in the class definition creates attributes of the class.

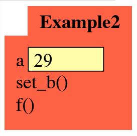
Every method has a special first parameter self that receives a reference to the instance the method was called on.

```
class Example2(object):
    """A class that defines some things."""

    # This is a class variable.
    a = 29

    # This is a method that
    # writes to an instance variable.
    def set_b(self, x):
        self.b = x

    # This is a method that reads
    # from a class variable and an
    # instance variable.
    def f(self):
        return self.a * self.b
```

A variable that lives in a class is a *class variable*.

A function that lives in a class defines a *method*.

This assignment will create an *instance variable*.



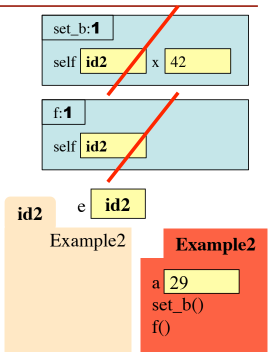
Method calls

Given class definition from previous slide:

`e = Example2()`
- constructor expression assigned to e
- creates a new instance, stores ID in e

`e.set_b(42)`
- method call has object + one argument
- turns into function call with 2 arguments
- value of e passed to self; 42 passed to x
- assignment to self.b creates instance var.

`print e.f()`
- method call has object + no arguments
- turns into function call with 1 arguments
- value of e passed to self
- attribute references find self.a in class, self.b in instance



Initializing instances

- Instances are initially empty.
- Usually we want to immediately add some instance variables.
- To make this easy, Python will automatically call a method named __init__ (if you declared one) right after creating an object, before the constructor call returns.

```
class Worker(object):
    """An instance is a worker in a
    certain organization.
    Instances have these variables:
        lname [string]: Last name
        ssn [int]: Social security
        boss [Worker]: Immediate boss
    """
    def __init__(self, lname, ssn, boss):
        self.lname = lname
        self.ssn = ssn
        self.boss = boss
```

gives access to the instance being initialized

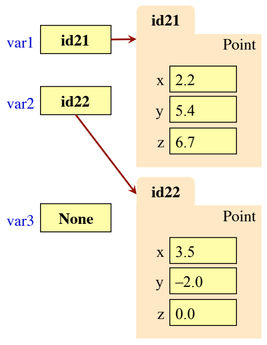
note **two** underscores

```
w = Worker("Obama", 1234, None)
```

this statement creates a new Worker instance, calls __init__ to set it up, and stores the name into w.

Aside: The value None

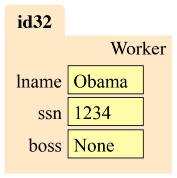
- The boss field is a problem.
  - boss is supposed to refer to a Worker object
  - But some workers might not have a boss
  - Maybe not assigned yet, maybe the buck stops there.
- **Solution**: use value None
  - **None**: Lack of (folder) name
  - Will reassign the field later!
- Be careful with None variables
  - var3.x gives error!
  - There is no name in var3
  - Which Point to use?



Evaluating a Constructor Expression

Worker('Obama', 1234, None)

1. Create a new object (folder) that is an instance of the class
   - Instance is initially empty
2. Call the method __init__ (if it exists)
   - Pass folder ID to self
   - Pass other arguments in order
3. 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 [float]: x coord
    y [float]: y coord
    z [float]: z coord"""

    def __init__(self, x=0, y=0, z=0):
        self.x = float(x)
        self.y = float(y)
        self.z = float(z)
    ...
```

What does str() do on class 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 called __str__
- With the defns. on these slides:

```
print Point(3,4,5)
```

produces the output:

```
(3.0,4.0,5.0)
```

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

Important!

YES	NO
`class Point(object):`	`class Point:`
`"""Instances are 3D points`	`"""Instances are 3D points`
`x [float]: x coord`	`x [float]: x coord`
`y [float]: y coord`	`y [float]: y coord`
`z [float]: z coord"""`	`z [float]: z coord"""`
`...`	`...`
3.0-Style Classes Well-designed	"Classic" Classes No reason to use these