CS1 Lecture 25 Mar. 22, 2017

- HW6 available this afternoon, due next Wed.
- No discussion sections this week

Last time

- Classes/object-oriented programming intro and initial example.

Today

- More classes and object-oriented programming. Chs 15, 16, 17

(last time) Ch15-18. Classes and Object-oriented (OO) programming

- A very important topic for modern programming.
  - Many many real-world systems are heavily object-oriented. E.g. to program iOS/iPhone/iPad, you'll have to deal with large complex OO libraries/frameworks
- It's a very big topic.
  - terms like: class, object, method, instance, inheritance, abstraction, encapsulation, information hiding, polymorphism, …
  - we'll cover the basics

(last time) Introduction to Classes

- defs lets us add new functions. Extremely useful for breaking down large program into components, building modules or libraries of computational tools
- classes let us define whole new types. Think of a class as a set of objects (the *instances* of the class) and the operations defined on them.
  - You are now familiar with: int, float, Boolean, string, list, tuple, dictionary
  - **with class definitions you can create your own types**. Programs can be much clearer, easier to understand and maintain when written in terms of appropriate types and instances of those types

Instead of using, say, a list or dictionary to represent a person:

```
p = ['jim', 55, 'blue', 'professor']
p[1]                # access age
```

and using basic list operations to extract age, define and use a Person class and related operations

```
p = Person(…)
p.age()
p.eyecolor()
p.occupation()
```

(last time) Classes

Basic python types are actually themselves classes.

- list objects are instances of the list class
- the operations defined for a class are called methods.

You've been using methods via the dot notation:

[1,2,3].append(4)

- Earlier I suggested you think about such methods as strange function call syntax [1,2,3].append(4) → append([1,2,3],4)

Methods *are indeed* functions – just special ones specific to a class. E.g. the list append method is defined *as part of* the definition of the list class.

- Execute help(list) in Python shell to see things defined for the list class

Defining classes

- In Python (and other languages) to define a **class**, you define object **attributes** (also often called **properties**) and the methods (operations) that can be invoked on objects (instances) ofthat class. General form:

```
class Myclass ():
    classAttribute1 = …
    …

    def method1(self, …):
        self.objectAttribute1 = …
        self.objectAttribute2 = …
        … computation in terms of properties and arguments passed to method…
        return …

    def method2(self, …)
        … computation in terms of properties and arguments passed to method ...
```

- Note: variable name self is a convention (standard practice/usage). The first argument to a method is always the object that invoked the method. It is *legal* to name it anything but please stick to standard practive – use 'self'

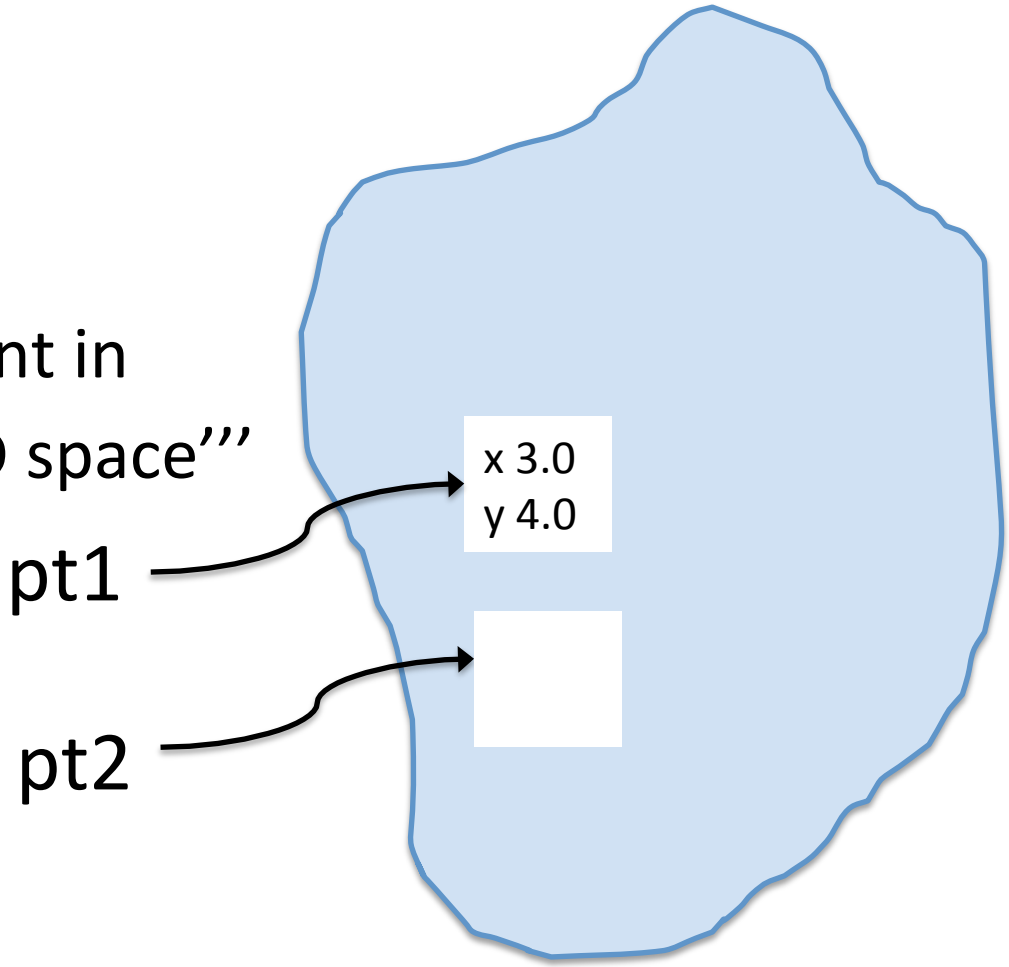
Last time

- Basic example to see how it works: Cat, Dog classes
  - Simple attributes and methods
  - Initialization via __init__ method
  - Looping over objects calling methods of same name
  - Nice, informative print form

Chapter 15: demonstrates classes as simple containers of attributes, but without methods. (*Many people would not really call this "object-oriented programming)*

E.g. >>> class Point:
'''represents a point in 2D space'''

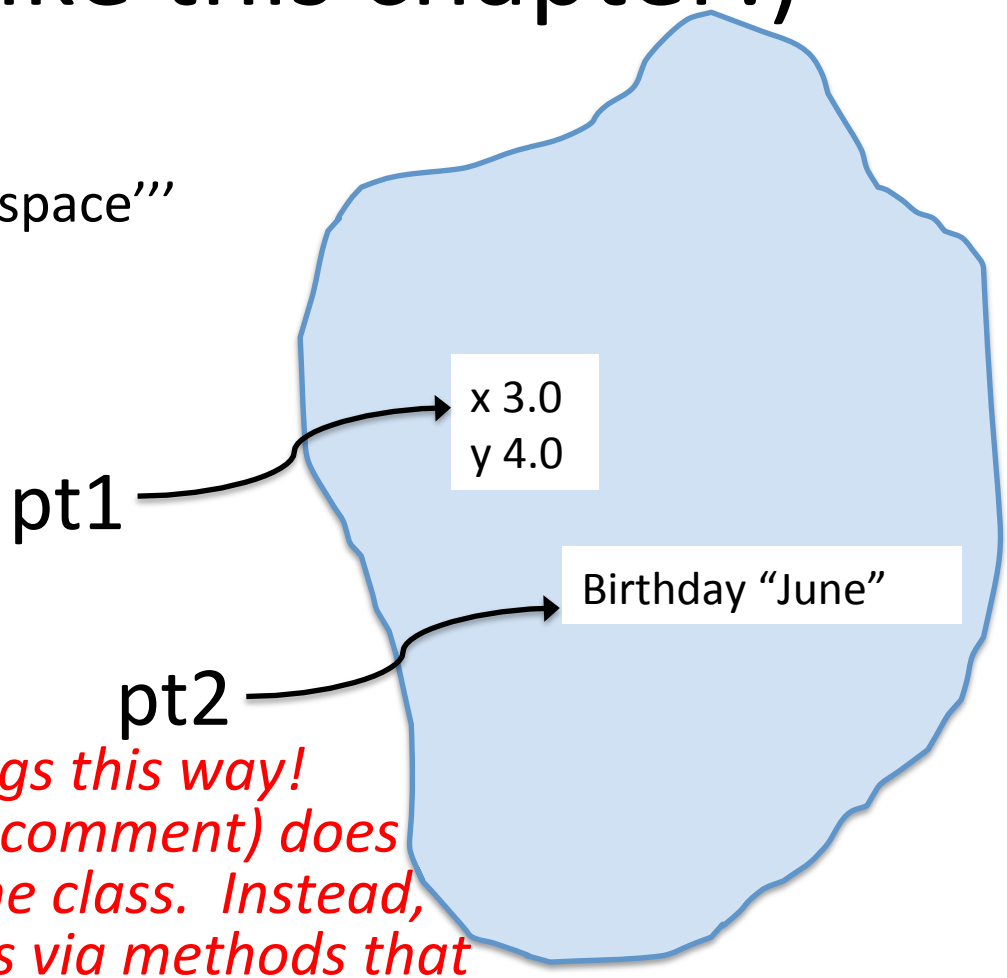
```
>>> pt1 = Point()
>>> pt1.x = 3.0
>>> pt1.y = 4.0
>>> pt2 = Point()
```



Ch 15 (I don't like this chapter!)

E.g. >>> class Point:
'''represents a point in
2D space'''

```
>>> pt1 = Point()
>>> pt1.x = 3.0
>>> pt1.y = 4.0
>>> pt2 = Point()
>>> pt2.birthday = "June" ???
```

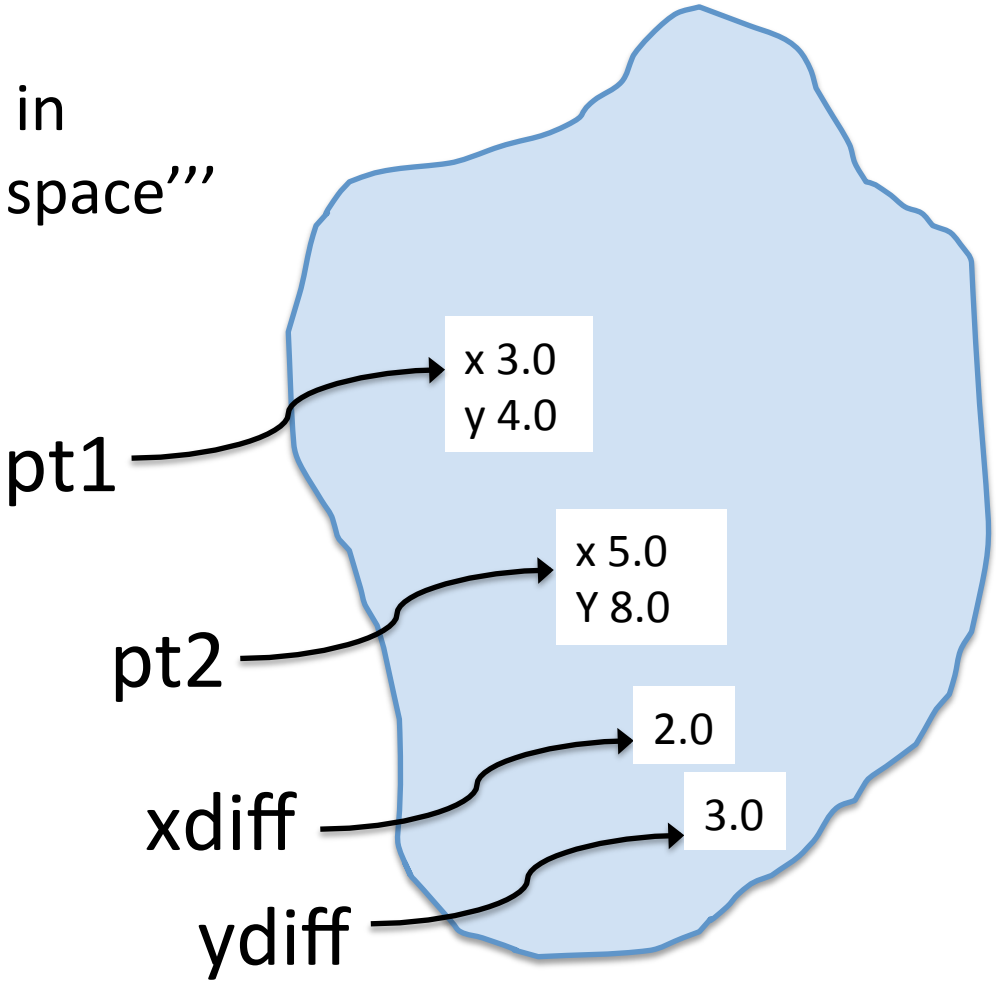


We usually don't want to do things this way! An empty class definition (plus a comment) does not define the interface/API to the class. Instead, good practice is to define a class via methods that are used to work with instances of the class. And, though legal, it is good practice ***not*** *to access attributes directly (pt1.x) but only via methods ("getters" and "setters")*

Ch 15

E.g. >>> class Point:
'''represents a point in
2D space'''

```
>>> pt1 = Point()
>>> pt1.x = 3.0
>>> pt1.y = 4.0
>>> pt2 = Point()
>>> pt2.x = 5.0
>>> pt2.y = 8.0
>>> xdiff = pt2.x – pt1.x
>>> ydiff = pt2.y – pt1.y
```



But, it is important to know how to access attributes since the methods you write in class definitions will access attributes directly.

Ch 16 – using classes in functions

- This is another chapter that, like 15, is not really demonstrating "object-oriented" style. It demonstrates use of classes as containers for sets of properties.
  - It mainly contains some examples of functions that do things with instances of user-defined classes like those in Ch15 *that have no methods*.
- Again, we don't usually want to do things this way – writing top-level functions that access object attributes directly (point.x, etc.)
- HOWEVER, it can be useful
- Will demonstrate a Time class using this approach and then see how we can convert that to the more standard object-oriented style of Ch 17.

ch16time.py

Ch 17

- This is the chapter to study most carefully
- General rule for defining classes:
  - define an init method initializing values for all properties/attributes (e.g. hour, minutes, seconds for time)
  - define methods that represent the "public interface" to the class. Users should work with instances of the class only via these methods rather than by accessing object attributes directly

ch17time.py – ch16time.py done in the recommended/more standard object-oriented style

Next time

Ch 17 next time, and start important idea of Inheritance (Ch 18)