Lect 10 – Objects

Rob Capra

INLS 490-172

Object-Oriented Programming

- OOP is a different way of thinking about writing programs
- Procedural:
  - cook(oven)
  - I have a cook() procedure that I pass oven to
- Object-oriented:
  - oven.cook()
  - I have an oven object that I ask to invoke its cook method

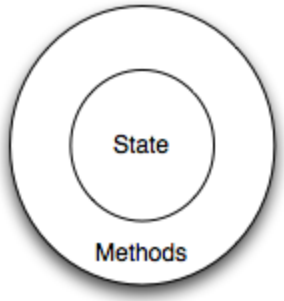
Object-Oriented Programming

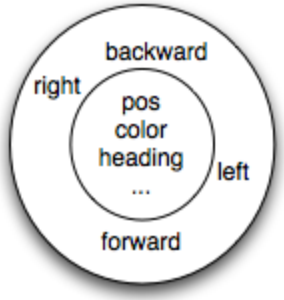
- Advantages
  - Provides an organization for our code, functions, methods
  - This organization can mirror real-world objects
  - Is often a logical way to think about our software architecture
  - Objects can be self-contained – methods you need are part of the object

Objects

- In Python, every value is an object
  - Integers, lists, dict, turtles – are all objects
- Programs manipulate objects in two ways
  - Perform some computation with the object
  - Ask the object to perform one of its methods

Objects

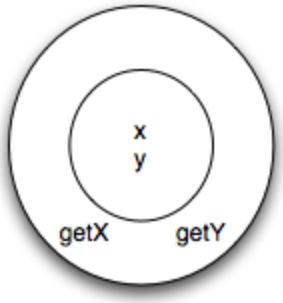




- Objects have:
  - State – what the object knows about itself
    - E.g., position, color, capacity
  - Methods – actions that it can perform
    - E.g., move, rotate, simplify, length
- Think about objects we have seen so far
  - String, list, dict, turtle
  - What kinds of state information and methods do they have?

Example #1 – Point

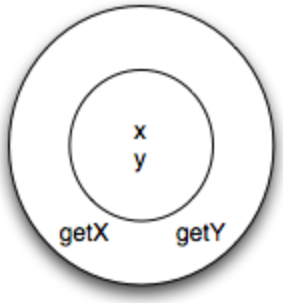
- Point class – represent x,y coordinates




```
class Point:
    """ Point class for representing x,y coordinates. """
    def __init__(self):
        """ Create a new point at 0,0 """
        self.x = 0
        self.y = 0

p = Point()
q = Point()
```

Example #1 – Point

- Point class – represent x,y coordinates

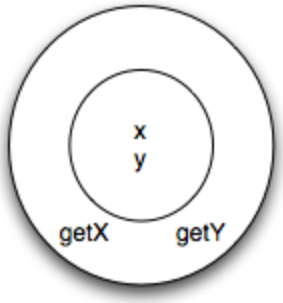



```
class Point:
    """ Point class for representing x,y coordinates. """
    def __init__(self):
        """ Create a new point at 0,0 """
        self.x = 0
        self.y = 0

p = Point()
q = Point()
```

Initializer method
(also called the constructor)
Called every time we create a new instance of Point

Example #1 – Point

- Point class – represent x,y coordinates

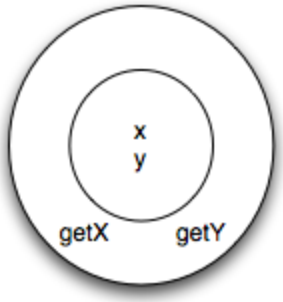



```
class Point:
    """ Point class for representing x,y coordinates. """
    def __init__(self):
        """ Create a new point at 0,0 """
        self.x = 0
        self.y = 0

p = Point()
q = Point()
```

`self` is automatically set to reference the new object
`self.x` is an ***attribute*** that is being set for the object
`self.y` is another ***attribute*** of the object

Example #1 – Point

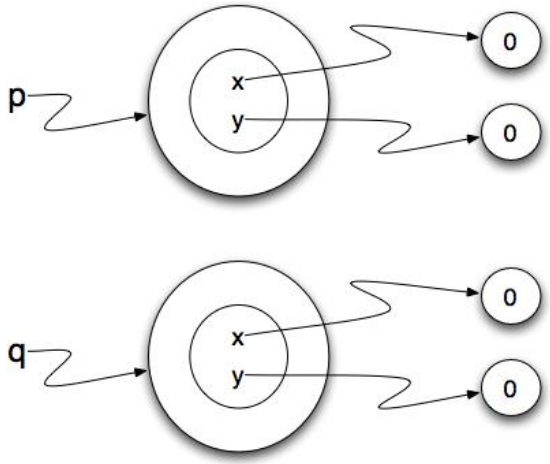
- Point class – represent x,y coordinates




```
class Point:
    """ Point class for representing x,y coordinates. """
    def __init__(self):
        """ Create a new point at 0,0 """
        self.x = 0
        self.y = 0
```

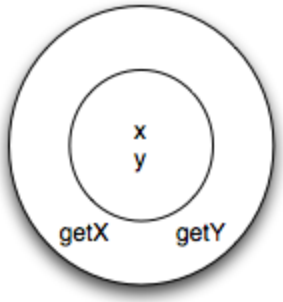
```
p = Point()
q = Point()
```

These lines create two new instances of Point



Example #1 – Point

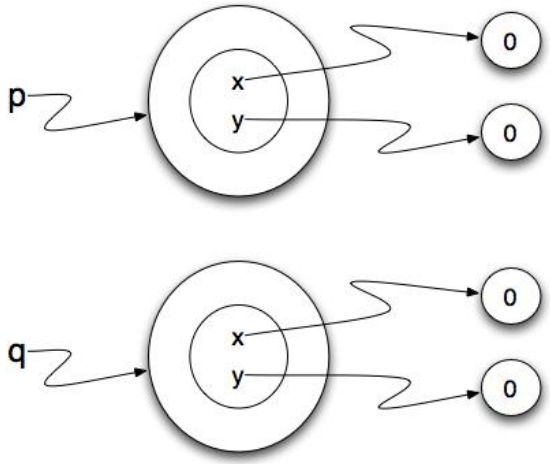
- Point class – represent x,y coordinates




```
class Point:
    """ Point class for representing x,y coordinates. """
    def __init__(self):
        """ Create a new point at 0,0 """
        self.x = 0
        self.y = 0

p = Point()
q = Point()

print p
print q

print (p is q)
```

Think of the class as a factory for making instances of the object.

As objects leave the factory, the __init__ method is called so that the object gets the correct default settings.

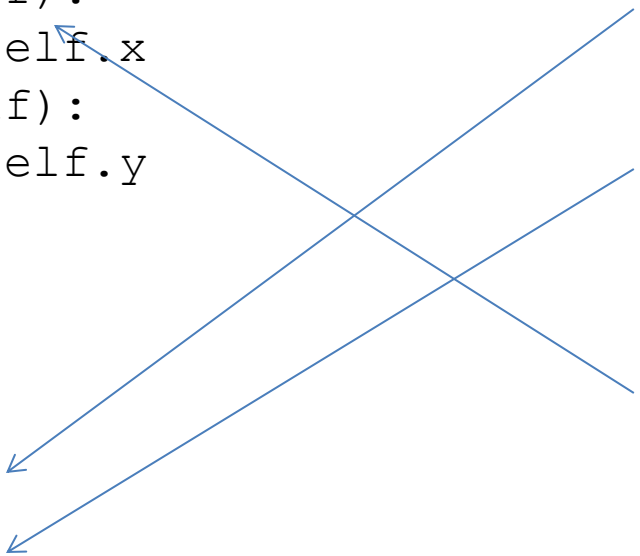


Improving the constructor

```
class Point:
    def __init__(self, initX, initY):
        self.x = initX
        self.y = initY

p = Point(3,4)
q = Point(5,6)

print p
print q
```

Adding methods to the class

```
class Point:
    def __init__(self, initX, initY):
        self.x = initX
        self.y = initY
    def getX(self):
        return self.x
    def getY(self):
        return self.y

p = Point(3,4)
q = Point(5,6)

print p.getX()
print p.getY()
print q.getX()
print q.getY()
```



Methods are like functions, but act on a specific instance of a class.

Methods are accessed using the dot notation.

Notice that we do not pass any other parameters to getX.

All methods that operate on objects will have **`self`** as their first parameter. This is a reference to the object itself, which allows the method access to the state data inside the object (e.g. the attributes).

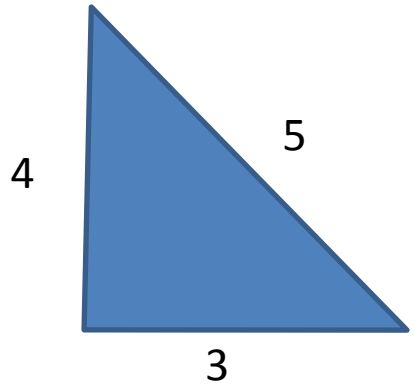
Remember Pythagoras?

```
class Point:
    def __init__(self, initX, initY):
        self.x = initX
        self.y = initY
    def getX(self):
        return self.x
    def getY(self):
        return self.y
    def distFromOrig(self):
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5

p = Point(3,4)

print p.getX()
print p.getY()
print p.distFromOrig()
```

$$a^2 + b^2 = c^2$$

$$c = \sqrt{a^2 + b^2}$$

$$c = (a^2 + b^2)^{0.5}$$



Objects as Arguments

```
import math

class Point:
    def __init__(self, initX, initY):
        self.x = initX
        self.y = initY
    def getX(self):
        return self.x
    def getY(self):
        return self.y

def distance(point1, point2):
    xdiff = point2.getX()-point1.getX()
    ydiff = point2.getY()-point1.getY()

    dist = math.sqrt(xdiff**2 + ydiff**2)
    return dist

p = Point(3,4)
q = Point(0,0)

print distance(p,q)
```

Distance is a function, but is NOT a method of Point.

Distance takes two Points as arguments and returns a number.

Special methods: __str__

```
class Point:
    def __init__(self, initX, initY):
        self.x = initX
        self.y = initY
    def getX(self):
        return self.x
    def getY(self):
        return self.y
    def __str__(self):
        return "x=" + str(self.x) + ", y=" + str(self.y)

p = Point(3,4)

print p
```

__str__ is a special method

Python will call this method on an object when you try to print it.

Previously, Python was using a default __str__ method for the object.

We **override** this method by defining our own __str__.

Returning Objects

```
class Point:
    def __init__(self, initX, initY):
        self.x = initX
        self.y = initY
    def getX(self):
        return self.x
    def getY(self):
        return self.y
    def __str__(self):
        return "x=" + str(self.x) + ", y=" + str(self.y)
    def halfway(self, target):
        mx = (self.x + target.x)/2
        my = (self.y + target.y)/2
        return Point(mx,my)

p = Point(3,4)
q = Point(5,12)
mid = p.halfway(q)
print mid
print mid.getX()
print p.halfway(q).getX()
```

Since halfway returns a new Point, we can use it just like any other Point.
For example, we can invoke its methods.