Classes and Objects

User-defined Types

- In addition to creating *functions*, Python lets us create our own *types* known as *classes.*
- The following defines a *Point* as a type of ***object.***

```
class Point(object):
    """represents a point in 2-D
space"""
```


User-defined Types

- Defining a class named *Point* creates a class object.

```
class Point(object):
    """represents a point in 2-D
space"""

>>> print Point
<class '__main__.Point'>
```


Instantiation

- A class object is like a factory for creating objects – in this case *Points.*
- Creating objects is called *instantiation* and the object created as *instances.*

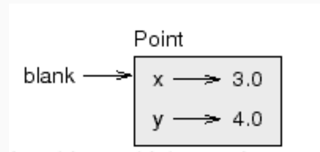
```
class Point(object):
    """represents a point in 2-D space"""

>>> point = Point()
>>> print point
<__main__.Point instance at 0xb7e9d3ac>
```

Attributes

• Objects can have their own data elements called *attributes.*

• Dot notation can be used to assign values to attributes.

```
>>> blank = Point()
>>> blank.x = 3.0
>>> blank.y = 4.0
>>> print blank.x
3.0
```



Using Dot Notation

• Dot notation can be used just like any other expression.

```
>>> distance = math.sqrt(blank.x**2 + blank.y**2)
>>> print distance
5.0
```


Passing an Object to a Function

• Objects can be passed into functions.

```
def print_point(p):
    print '(%g, %g)' % (p.x, p.y)

>>> print_point(blank)
(3.0, 4.0)
```


A Distance Function

• Write a function that returns the distance between two points passed as parameters.

```
def distance(p1, p2):
  '''Returns the distance between Points p1 and p2'''
  return math.sqrt((p2.x-p1.x)**2 + (p2.y-p1.y)**2)
```

A Distance Function

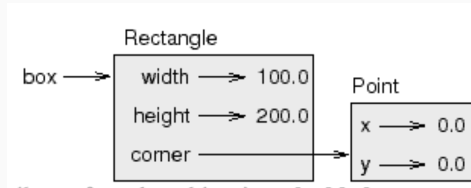
- Find the distance between blank (3.0, 4.0) and the origin.

```
>>>  origin = Point()
>>> origin.x = 0
>>> origin.y = 0
>>> print 'distance between blank and origin  =
', distance(blank, origin)
```


A Rectangle Object

- Represented as width, height, and one corner.

```
class Rectangle(object):
    """represent a rectangle.
       attributes: width, height, corner.
    """
box = Rectangle()
box.width = 100.0
box.height = 200.0
box.corner = Point()
box.corner.x = 0.0
box.corner.y = 0.0
```



Area of Rectangle

- Write a function that returns the area of a rectangle.

```
def area(rect):
   ''' returns area of the rectangle, rect'''
   return rect.width * rect.height

>>> print 'the area of box = ', area(box)
the area of box =  20000.0
```


Instances as Return Values

- Functions can return instances as their return values.

```
def find_center(rect):
    ''' Returns the center of a rectangle'''
    p = Point()
    p.x = rect.corner.x + rect.width/2.0
    p.y = rect.corner.y + rect.height/2.0
    return p
```

Instances as Return Values

- Functions can return instances as their return values.

```
>>> center = find_center(box)
>>> print_point(center)
(50.0, 100.0)
```


Objects are Mutable

- An object's attributes can be changed.

```
box.width = box.width + 50
box.height = box.width + 100
```


Functions and Aliasing

- A object parameter creates an alias.

```
def grow_rectangle(rect, dwidth, dheight) :
    rect.width += dwidth
    rect.height += dheight
>>> print box.width
100.0
>>> print box.height
200.0
>>> grow_rectangle(box, 50, 100)
>>> print box.width
```


Functions and Aliasing

- A object parameter creates an alias.

```
def grow_rectangle(rect, dwidth, dheight) :
    rect.width += dwidth
    rect.height += dheight
>>> print box.width
100.0
>>> print box.height
200.0
>>> grow_rectangle(box, 50, 100)
>>> print box.width
150.0
```

Functions and Aliasing

- A object parameter creates an alias.

```
def grow_rectangle(rect, dwidth, dheight) :
    rect.width += dwidth
    rect.height += dheight
>>> print box.width
100.0
>>> print box.height
200.0
>>> grow_rectangle(box, 50, 100)
>>> print box.width
150.0
>>> print box.height
```


Functions and Aliasing

- A object parameter creates an alias.

```
def grow_rectangle(rect, dwidth, dheight) :
    rect.width += dwidth
    rect.height += dheight
>>> print box.width
100.0
>>> print box.height
200.0
>>> grow_rectangle(box, 50, 100)
>>> print box.width
150.0
>>> print box.height
300.0
```


Copying vs. Aliasing

- Copying an object (two objects, two names) is an alternative to aliasing (one object, two names).
- Here, p1 and p2 will point to two different objects that have the same x, y values.

```
>>> p1 = Point()
>>> p1.x = 3.0
>>> p1.y = 4.0

>>> import copy
>>> p2 = copy.copy(p1)
```


Copying vs. Aliasing

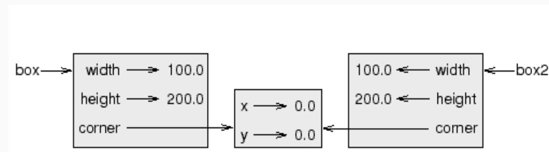
- Now, p1 and p2 point to two different objects that have the same x, y values.

```
>>> print_point(p1)
(3.0, 4.0)
>>> print_point(p2)
(3.0, 4.0)
>>> p1 is p2
False
>>> p1 == p2  # Default: in and == are the same
False
```

Shallow Copy

- The copy.copy() method performs a *shallow* copy – it does NOT copy embedded objects.

```
>>> box2 = copy.copy(box)
>>> box2 is box
False
>>> box2.corner is box.corner
True
```



Deep Copy

- The copy.deepcopy() method performs a *deep* copy – it DOES copy embedded objects.

```
>>> box3 = copy.deepcopy(box)
>>> box3 is box
False
>>> box3.corner is box.corner
False
```


Inclass Exercises

- Write a function named move_rectangle that takes a Rectangle and two numbers named dx and dy. It should change the location of the rectangle by adding dx to the x coordinate of corner and adding dy to the y coordinate of corner.
- Write a version of move_rectangle that creates and returns a new Rectangle instead of modifying the old one.