Think Python 2e, Chapter 15 Notes

Classes and Objects

October 20, 2022

A new data type: point

To represent a two dimensional point in the plane, (x, y), we could:

- Use two variables, `x` and `y`
- Use a list or tuple.
- Use a dictionary with `'x'` and `'y'` as keys.
- Create a new data type to represent points as objects.

A new data type

```
class Point:
    """Represents a point in 2-D space."""
```

- Defining a class named Point creates a **class object**.
- The string is the documentation string, or **docstring**.
- Class names are traditionally capitalized.

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

- Because Point is defined at the top level, its "full name" is `__main__.Point`.

A new data type: Point

```
1 class Point:
2     """Represents a point in 2-D space."""
```

- The class object is like a factory for creating objects.
- To create a Point, you call `Point` as if it were a function.

```
1 >>> blank = Point()
2 >>> blank
3 <__main__.Point object at 0xb7e9d3ac>
```

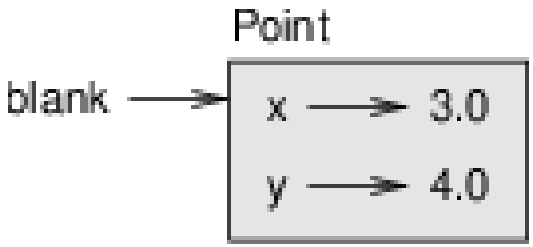
- The return value is a reference to a `Point` object.
- Creating a new object is called **instantiation**.
- The object is an **instance** of the class.
- The hex number is its location in memory.
- In Python every object is an instance of some class.

Attributes

User-defined objects can have **attributes**.

```
>>> blank.x = 3.0
>>> blank.y = 4.0
>>> '(%g, %g)' % (blank.x, blank.y)
'(3.0, 4.0)'
>>> distance = math.sqrt(blank.x**2 + blank.y**2)
>>> distance
5.0
```

State diagram:



Functions of objects

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

```
1 >>> print_point(blank)
2 (3.0, 4.0)
```

Objects are aliased, so if the function modifies `p`'s attributes, `blank`'s attributes change, too.

A new data type: Rectangle

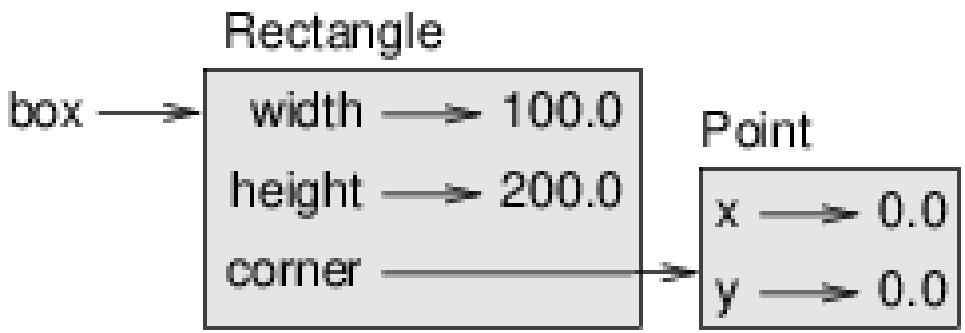
```
class Rectangle:
    """Represents a rectangle.

    attributes: width, height, corner.
    """
```

Are there other good representations?
What about tilted rectangles?

A new data type: Rectangle

```
box = Rectangle()
box.width = 100.0
box.height = 200.0
box.corner = Point()
box.corner.x = 0.0
box.corner.y = 0.0
```



State diagram:

An object that is an attribute of another object is **embedded**.

Instances as return values

```
def find_center(rect):
    p = Point()
    p.x = rect.corner.x + rect.width/2
    p.y = rect.corner.y + rect.height/2
    return p
```

```
>>> center = find_center(box)
>>> print_point(center)
(50, 100)
```

Objects are mutable

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

```
def grow_rectangle(rect, dwidth, dheight):
    rect.width += dwidth
    rect.height += dheight
```

```
>>> box.width, box.height
(150.0, 300.0)
>>> grow_rectangle(box, 50, 100)
>>> box.width, box.height
(200.0, 400.0)
```

Copying

- Aliasing with objects can be problematic.
- The copy module can copy anything.

```
>>> p1 = Point()
>>> p1.x = 3.0
>>> p1.y = 4.0
>>> import copy
>>> p2 = copy.copy(p1)
>>> print_point(p1)
(3, 4)
>>> print_point(p2)
(3, 4)
>>> p1 is p2
False
>>> p1 == p2
False
```

Copying

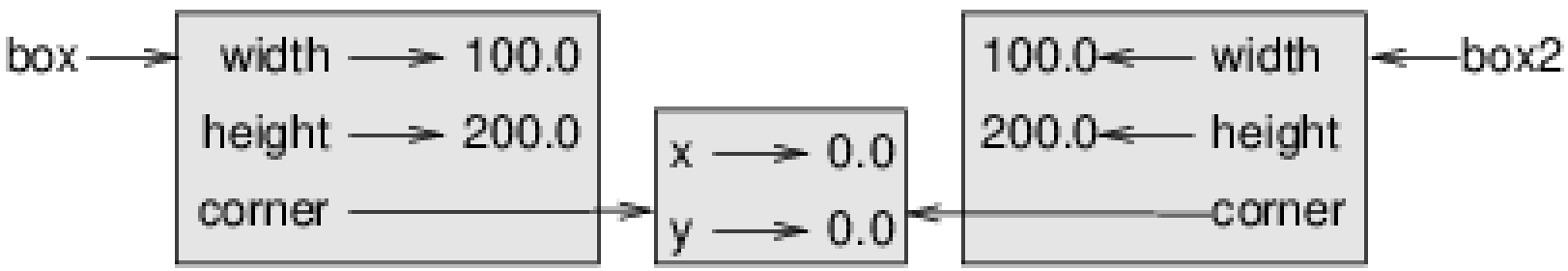
```
>>> print_point(p1)
(3, 4)
>>> print_point(p2)
(3, 4)
>>> p1 is p2
False
>>> p1 == p2
False
```

- Python does not presume to know what counts as ==
- But you can tell it! (More on this later)

Shallow copy

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

Object diagram:



Deep copy

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

What does the object diagram look like?

Vocabulary

class: A programmer-defined type. A class definition creates a new class object.

class object: An object that contains information about a programmer-defined type. The class object can be used to create instances of the type.

instance: An object that belongs to a class.

instantiate: To create a new object.

attribute: One of the named values associated with an object.

embedded object: An object that is stored as an attribute of another object.

Vocabulary

shallow copy: To copy the contents of an object, including any references to embedded objects; implemented by the copy function in the copy module.

deep copy: To copy the contents of an object as well as any embedded objects, and any objects embedded in them, and so on; implemented by the deepcopy function in the copy module.

object diagram: A diagram that shows objects, their attributes, and the values of the attributes.