Lab 5:

Class and object:

Class : A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

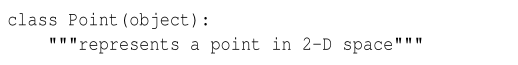
Class Variable :A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.

Instance Variable: A variable that is defined inside a method and belongs only to the current instance of a class.

A class definition looks like this:

class **name**:

**statements**

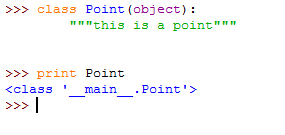


You need to put class keyword first and the name of your class. In this example, the header indicates the class is a Point.

The body is a docstring that explains what the class is for. You can define variables and functions inside a class definition.

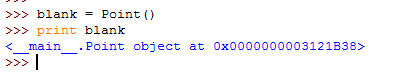
In above example:

Defining a class named Point creates a class object.



Because Point is defined at the top level, its full name is \_\_main\_\_.Point.

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

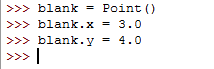


The return value is reference to a Point object, which we assign to blank. Creating a new object is called instantiation, and the object is an instance of the class.

When you print an instance, Python tells you what class it belongs to and where it is stored in memory.

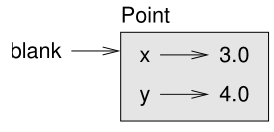
Attributes:

We can assign values to an instance using dot notation.

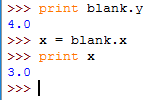


The x and y elements of an object blank is called attributes.

The following diagram shows the result of these assignment. A state diagram that shows an object and its attributes is called object diagram.



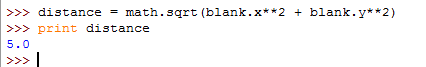
The variable blank refers to a Point object, which contains two attributes. Each attribute refers to a floating-point number.



The expression blank.x means, “Go to the object blank refers to and get the value of x.

You can use dot notation as part of any expression. For example:





You can pass an instance as an argument in the usual way. Example below:

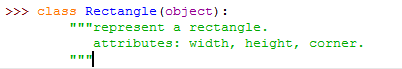


The print\_point takes a point as an argument and displays it in a mathematical notation.

Inside the function, p is an alias for blank , so if the function modifies p , blank changes.

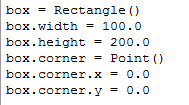
Rectangles:

Lets define the class Rectangle specifying one corner of the rectangle, the width, and the height.



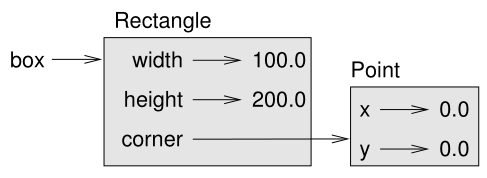
The docstring lists the attributes: width and height are numbers; corner is a Point object that specifies the lower-left corner.

To represent a rectangle, youhavetoinstantiateaRectangleobjectandassignvaluestotheattributes:



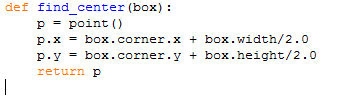
The expression box.corner.x means, “Go to the object box refers to and select the attribute named corner ; then go to that object and select the attribute named x .”

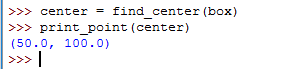
The state diagram of this object is:



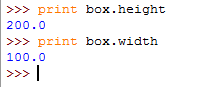
Instances as return values:

Functions can return instances. For example, find\_center takes a Rectangle as an argument and returns a Point that contains the coordinates of the center of the Rectangle:



the following example passes a box as an argument and assigns the resulting Point to center:

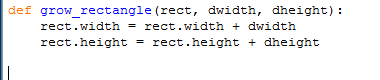
objects are immutable:

You can change the state of an object by making an assignment to one of its attributes. For example, to change the size of a rectangle without changing its position, you can modify the values of width and height:

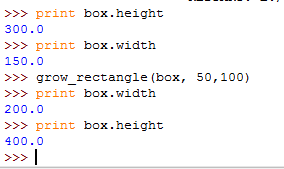


Result🡪

You can also write functions that modify objects. For example, grow\_rectangle takes a Rectangle object and two numbers, dwidth and dheight , and adds the numbers to the width and height of the rectangle:



Here is an example that demonstrates the effect:

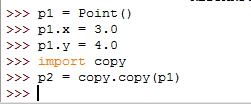


Inside the function rect is an alias for box, so if the function modifies rect, box changes.

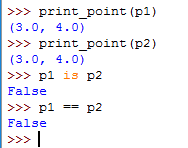
Copying:

Aliasing can make a program difficult to read because changes in one place might have unexpected effects in another place. It is hard to keep track of all the variables that might refer to a given object.

Copying an object is often an alternative to aliasing. The copy module contains a function called copy that can duplicate any object:

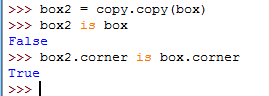


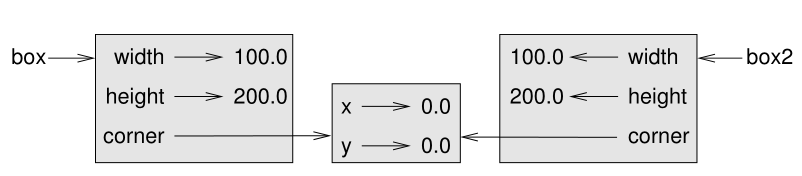
P1 and p2 contain the same data, but they are not the same Point.



The is operator indicates that p1 and p2 are not the same object, which is what we expected. But you might have expected == to yield True because these points contain the same data. In that case, you will be disappointed to learn that for instances, the default behavior of the == operator is the same as the is –

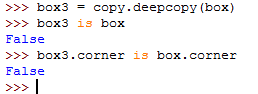
If you use copy.copy to duplicate a Rectangle, you will find that it copies the Rectangle object but not the embedded Point.



The object diagram look like this:

This operation is called a shallow copy because it copies the object and any references it contains, but not the embedded objects.

The copy module contains a method named deep copy that copies not only the object but also the objects it refers to, and the objects they refer to, and so on. You will not be surprised to learn that this operation is called a deep copy.



So box3 and box are completely separate objects.