We begin defining our own classes.

1. Questions? Recalling a subset iterator.

2. Beginning today, we develop our own type types or classes. Every value in Python is an object. The type of an object is its “species.” The class is a formal definition of a type.

3. Classes can be thought of as factories for making objects. The existence of a class does not imply the existence of any of its objects. (Consider the martian class.)

4. Objects have state, which is typically held in instance variables or (a very Pythonic term:) attributes. When we access objects, we often “ask” about their state. We could do this by inspecting their attributes.

5. A point (Pt), in Python.

6. To access the attributes of an object, we simply append them to the object with a dot (.).

7. We can restrict the construction of instance variables of an object by specifying the __slots__ attribute for the class: it’s a list of instance variables allowed for each class object.

8. Classes have an initializer, __init__, that describes how the class sets the attributes of a new object of this type.

9. The initializer is an example of a method, an object-specific function. As is true with methods generally, within the initializer, the parameter self is used to refer to the object at hand. The parameter self is always the first parameter and never directly specified. In the case of an initializer, self is simply the object produced by a call to the class, here Pt().

10. Other methods of the class fall in two broad categories: accessor methods that give us read-only access to the state of the object, and mutator methods that allow us to modify the state of an object.

11. Ideally, we don’t allow the user to have direct access to the state of the object. Instead, we control access to the state through methods.

12. Used effectively, this approach directly supports data abstraction: the methods provide the public interface, while the attributes are features of the private implementation.

13. Definition of a rectangle (Rect).

*