More Examples of Classes

Announcements

- Hand in written part of lab 10
- Lab 10 programs due tomorrow at midnight
  - Ninja session tonight
- Quiz 6 (last quiz before final) next Friday
  - Will focus on sorting and recursion

Plan

- Review how to define a class
- Reminder that objects are mutable
- Reasons why we’d define a class
- More examples of classes

Review

- A **class** is a template for a group of objects. You can think of it like we're adding a new type to Python.

- A class definition is comprised of **method** definitions. Every class needs at least an `__init__` method and a `__str__` method.

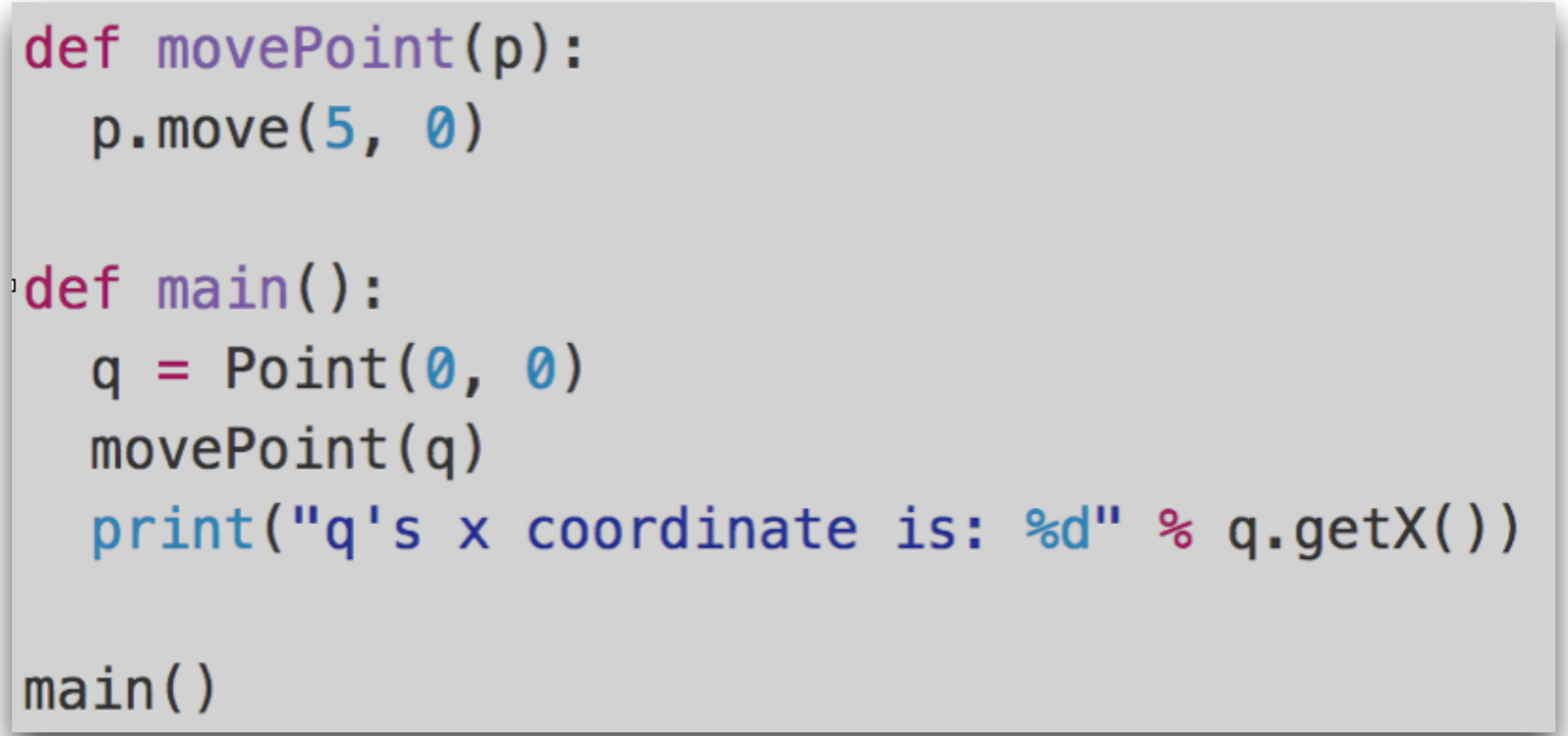
- The methods define an **interface** for how the object can be used.

Review

- You can create many **instances** of a class. Each one has its own set of **instance variables.** This is the information that's *private* to the object.
- Methods can access **instance variables**; they can also take parameters.
- Every method has `self` as its default first parameter.

```python
class Point(object):

  def __init__(self, x, y):
    self.x = x
    self.y = y

  def __str__(self):
    return "Point at (%d, %d)" % (self.x, self.y)

  def getX(self):
    return self.x

  def getY(self):
    return self.y

  def move(self, dx, dy):
    self.x = self.x + dx
    self.y += dy
```

Objects are mutable



```
def movePoint(p):
  p.move(5, 0)

def main():
  q = Point(0, 0)
  movePoint(q)
  print("q's x coordinate is: %d" % q.getX())

main()
```

Aside: exception handling

- We have seen how certain **runtime errors** or **exceptions** cause a program to crash. For instance, trying to use the `int` conversion function on a string that doesn’t contain an int.
- Sometimes we want to handle exceptions more gracefully. In these situation we can use Python’s `try-except` construct.

Why use classes?

- Like functions, classes allow us to split a program into more manageable pieces. This facilitates code reuse and collaboration.
- Like a function, a class is an **interface** that hides implementation details.
- Like functions, classes allow us to extend Python’s capabilities.

Why use classes?

- When we call a function we must supply all the needed information. When we use an object, it stores or “remembers” some information for us.

- Breaking a program down into classes sometimes meshes with the way we think about the world. This idea is called **object-oriented programming**.

    - e.g. a Course object with a list of Student objects, and a list of Date objects for when the course meets.

- Classes can standardize the way we store various kinds of data.

Why define our own classes?

- Because we want to write a program in some new domain for which there are no existing classes.
- Because we think an existing class should be designed differently.
- Because we want to facilitate collaboration by creating new interfaces.

How to define a class

- As you're doing a top-down design, create a wish list of the methods you need for the class. This is the interface to the class.

- Ask yourself: what data does the object need to remember about itself in order to implement all these methods? These pieces of data become the instance variables. Don't make any assumptions about the order in which methods will be called.

- Decide on the type for each instance variable. If there is no appropriate type, you may want to define another class to be used by this class.

- Implement the class, one method at a time, testing as you go.

Examples