20.5. Adding Other Methods to a Class

The key advantage of using a class like `Point` rather than something like a simple tuple `(7, 6)` now becomes apparent. We can add methods to the `Point` class that are sensible operations for points. Had we chosen to use a tuple to represent the point, we would not have this capability. Creating a class like `Point` brings an exceptional amount of "organizational power" to our programs, and to our thinking. We can group together the sensible operations, and the kinds of data they apply to, and each instance of the class can have its own state.

A **method** behaves like a function but it is invoked on a specific instance. For example, with a list bound to variable L, `L.append(7)` calls the function append, with the list itself as the first parameter and 7 as the second parameter. Methods are accessed using dot notation. This is why `L.append(7)` has 2 parameters even though you may think it only has one: the list stored in the variable `L` is the first parameter value and 7 is the second.

Let's add two simple methods to allow a point to give us information about its state. The `getX` method, when invoked, will return the value of the x coordinate.

The implementation of this method is straight forward since we already know how to write functions that return values. One thing to notice is that even though the `getX` method does not need any other parameter information to do its work, there is still one formal parameter, `self`. As we stated earlier, all methods defined in a class that operate on objects of that class will have `self` as their first parameter. Again, this serves as a reference to the object itself which in turn gives access to the state data inside the object.

Save & Run   Original - 1 of 1   Show in CodeLens

```
class Point:
    """ Point class for representing and manipulating x,y coordinates. """

    def __init__(self, initX, initY):

        self.x = initX
        self.y = initY

    def getX(self):
        return self.x

    def getY(self):
        return self.y


```

```
7
6
```

Activity: 1 -- ActiveCode (chp13_classes4)

Note that the `getX` method simply returns the value of the instance variable x from the object self. In other words, the implementation of the method is to go to the state of the object itself and get the value of `x`. Likewise, the `getY` method looks almost the same.

Let's add another method, `distanceFromOrigin`, to see better how methods work. This method will again not need any additional information to do its work, beyond the data stored in the instance variables. It will perform a more complex task.

Save & Run   Original - 1 of 1   Show in CodeLens

```
class Point:
    """ Point class for representing and manipulating x,y coordinates. """

    def __init__(self, initX, initY):

        self.x = initX
        self.y = initY

    def getX(self):
        return self.x

    def getY(self):
        return self.y

    def distanceFromOrigin(self):
```

```
9.21954445729
```

Activity: 2 -- ActiveCode (chp13_classes5)

Notice that the call of `distanceFromOrigin` does not *explicitly* supply an argument to match the `self` parameter. This is true of all method calls. The definition will always seem to have one additional parameter as compared to the invocation.

Check Your Understanding

1. Create a class called `Animal` that accepts two numbers as inputs and assigns them respectively to two instance variables: `arms` and `legs`. Create an instance method called `limbs` that, when called, returns the total number of limbs the animal has. To the variable name `spider`, assign an instance of `Animal` that has 4 arms and 4 legs. Call the limbs method on the `spider` instance and save the result to the variable name `spidlimbs`.

Save & Run 5/15/2021, 12:06:14 PM - 2 of 2 Show in CodeLens

```
class Animal:
    def __init__(self, arms, legs):
        self.arms = arms
        self.legs = legs

    def limbs(self):
        return self.arms + self.legs

spider = Animal(4, 4)
spidlimbs = spider.limbs()
print(spidlimbs)
```

```
8
```

Activity: 3 -- ActiveCode (ac_chp13_classes_01)

Result	Actual Value	Expected Value	Notes
Pass	4	4	Testing that spider was assigned the correct number of arms.
Pass	4	4	Testing that spider was assigned the correct number of legs.
Pass	8	8	Testing that spidlimbs was assigned correctly.

You passed: 100.0% of the tests

You have attempted 4 of 3 activities on this page

20.4. Adding Parameters to the Constructor">

20.6. Objects as Arguments and Parameters">

lding Parameters to the Constructor">

✔ Completed

20.6. Objects as Arguments and Parameters">Next Section - 20.6. Objects as Arguments and Parameters

| Back to top