17. Classes and Objects - the Basics

17.1. Object-oriented programming

17.2. A change of perspective

17.3. Objects Revisited

17.4. User Defined Classes

17.5. Improving our Constructor

17.6. Adding Other Methods to our Class

17.7. Objects as Arguments and Parameters

17.8. Converting an Object to a String

17.9. Instances as Return Values

17.10. Glossary

17.11. Exercises

17.1. Object-oriented programming

Python is an object-oriented programming language. That means it provides features that support object-oriented programming (OOP).

Object-oriented programming has its roots in the 1960s, but it wasn’t until the mid 1980s that it became the main programming paradigm used in the creation of new software. It was developed as a way to handle the rapidly increasing size and complexity of software systems and to make it easier to modify these large and complex systems over time.

Up to now, some of the programs we have been writing use a procedural programming paradigm. In procedural programming the focus is on writing functions or procedures which operate on data. In object-oriented programming the focus is on the creation of objects which contain both data and functionality together. Usually, each object definition corresponds to some object or concept in the real world and the functions that operate on that object correspond to the ways real-world objects interact.

17.2. A change of perspective

Throughout the earlier chapters, we wrote functions and called them using a syntax such as drawCircle(tess). This suggests that the function is the active agent. It says something like, “Hey, drawCircle! Here’s a turtle object for you to use to draw with.”

In object-oriented programming, the objects are considered the active agents. For example, in our early introduction to turtles, we used an object-oriented style. We said tess.forward(100), which asks the turtle to move itself forward by the given number of steps. An invocation like tess.circle() says “Hey tess! Please use your circle method!”

This change in perspective is sometimes considered to be a more “polite” way to write programming instructions. However, it may not initially be obvious that it is useful. It turns out that often times shifting responsibility from the functions onto the objects makes it possible to write more versatile functions and makes it easier to maintain and reuse code.

The most important advantage of the object-oriented style is that it fits our mental chunking and real-life experience more accurately. In real life our cook method is part of our microwave oven — we don’t have a cook function sitting in the corner of the kitchen, into which we pass the microwave! Similarly, we use the cellphone’s own methods to send an sms, or to change its state to silent. The functionality of real-world objects tends to be tightly bound up inside the objects themselves. OOP allows us to accurately mirror this when we organize our programs.

17.3. Objects Revisited

In Python, every value is actually an object. Whether it be a turtle, a list, or even an integer, they are all objects. Programs manipulate those objects either by performing computation with them or by asking them to perform methods. To be more specific, we say that an object has a state and a collection of methods that it can perform. The state of an object represents those things that the object knows about itself. For example, as we have seen with turtle objects, each turtle has a state consisting of the turtle’s position, its color, its heading and so on. Each turtle also has the ability to go forward, backward, or turn right or left. Individual turtles are different in that even though they are all turtles, they differ in the specific values of the individual state attributes (maybe they are in a different location or have a different heading).

17.4. User Defined Classes

We’ve already seen classes like str, int, float and Turtle. These were defined by Python and made available for us to use. However, in many cases when we are solving problems we need to create data objects that are related to the problem we are trying to solve. We need to create our own classes.

As an example, consider the concept of a mathematical point. In two dimensions, a point is two numbers (coordinates) that are treated collectively as a single object. Points are often written in parentheses with a comma separating the coordinates. For example, (0, 0) represents the origin, and (x, y) represents the point x units to the right and y units up from the origin. This (x,y) is the state of the point.

Thinking about our diagram above, we could draw a point object as shown here.

A point has an x and a y

Some of the typical operations that one associates with points might be to ask the point for its x coordinate, getX, or to ask for its y coordinate, getY. You may also wish to calculate the distance of a point from the origin, or the distance of a point from another point, or find the midpoint between two points, or answer the question as to whether a point falls within a given rectangle or circle. We’ll shortly see how we can organize these together with the data.

A point also has methods

Now that we understand what a point object might look like, we can define a new class. We’ll want our points to each have an x and a y attribute, so our first class definition looks like this.

class Point:

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

def \_\_init\_\_(self):

""" Create a new point at the origin """

self.x = 0

self.y = 0

Class definitions can appear anywhere in a program, but they are usually near the beginning (after the import statements). The syntax rules for a class definition are the same as for other compound statements. There is a header which begins with the keyword, class, followed by the name of the class, and ending with a colon.

If the first line after the class header is a string, it becomes the docstring of the class, and will be recognized by various tools. (This is also the way docstrings work in functions.)

Every class should have a method with the special name \_\_init\_\_. This initializer method, often referred to as the constructor, is automatically called whenever a new instance of Point is created. It gives the programmer the opportunity to set up the attributes required within the new instance by giving them their initial state values. The self parameter (you could choose any other name, but nobody ever does!) is automatically set to reference the newly-created object that needs to be initialized.

So let’s use our new Point class now.

1

class Point:

2

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

3

​

4

def \_\_init\_\_(self):

5

""" Create a new point at the origin """

6

self.x = 0

7

self.y = 0

8

​

9

p = Point() # Instantiate an object of type Point

10

q = Point() # and make a second point

11

​

12

print("Nothing seems to have happened with the points")

13

​

Activity: 17.4.1 ActiveCode (chp13\_classes1)

During the initialization of the objects, we created two attributes called x and y for each, and gave them both the value 0.

Note

The assignments are not to x and y, but to self.x and self.y. The attributes x and y are always attached to a particular instance. The instance is always explicitly referenced with dot notation.

You will note that when you run the program, nothing happens. It turns out that this is not quite the case. In fact, two Points have been created, each having an x and y coordinate with value 0. However, because we have not asked the point to do anything, we don’t see any other result.

Simple object has state and methods

You can see this for yourself, via codelens:

1 class Point:

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

3

4 def \_\_init\_\_(self):

5 """ Create a new point at the origin """

6 self.x = 0

7 self.y = 0

8

9 p = Point() # Instantiate an object of type Point

10 q = Point() # and make a second point

11

12 print("Nothing seems to have happened with the points")

The following program adds a few print statements. You can see that the output suggests that each one is a Point object. However, notice that the is operator returns False meaning that they are different objects (we will have more to say about this in a later chapter).

1 class Point:

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

3

​4 def \_\_init\_\_(self):

5 """ Create a new point at the origin """

6 self.x = 0

7 self.y = 0

8

​9 p = Point() # Instantiate an object of type Point

10 q = Point() # and make a second point

11

​12 print(p)

13 print(q)

14

​15 print(p is q)

16

​

This should look familiar — we’ve used classes before to create more than one object:

from turtle import Turtle

tess = Turtle() # Instantiate objects of type Turtle

alex = Turtle()

The variables p and q are assigned references to two new Point objects. A function like Turtle or Point that creates a new object instance is called a constructor. Every class automatically uses the name of the class as the name of the constructor function. The definition of the constructor function is done when you write the \_\_init\_\_ function.

It may be helpful to think of a class as a factory for making objects. The class itself isn’t an instance of a point, but it contains the machinery to make point instances. Every time you call the constructor, you’re asking the factory to make you a new object. As the object comes off the production line, its initialization method is executed to get the object properly set up with its factory default settings.

The combined process of “make me a new object” and “get its settings initialized to the factory default settings” is called instantiation.

17.5. Improving our Constructor

Our constructor so far can only create points at location (0,0). To create a point at position (7, 6) requires that we provide some additional capability for the user to pass information to the constructor. Since constructors are simply specially named functions, we can use parameters (as we’ve seen before) to provide the specific information.

We can make our class constructor more general by putting extra parameters into the \_\_init\_\_ method, as shown in this codelens example.

1 class Point:

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

3

4 def \_\_init\_\_(self, initX, initY):

5 """ Create a new point at the given coordinates. """

6 self.x = initX

7 self.y = initY

8

9 p = Point(7, 6)

Now when we create new points, we supply the x and y coordinates as parameters. When the point is created, the values of initX and initY are assigned to the state of the object.

17.6. Adding Other Methods to our 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 simple 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 turtle named tess, tess.right(90) asks the tess object to perform its right method and turn 90 degrees. Methods are accessed using dot notation.

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 reference to the object itself which in turn gives access to the state data inside the object.

1

class Point:

2

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

3

​

4

def \_\_init\_\_(self, initX, initY):

5

""" Create a new point at the given coordinates. """

6

self.x = initX

7

self.y = initY

8

​

9

def getX(self):

10

return self.x

11

​

12

def getY(self):

13

return self.y

14

​

15

​

16

p = Point(7, 6)

17

print(p.getX())

18

print(p.getY())

19

​

Activity: 17.6.1 ActiveCode (chp13\_classes4)

Note that the getX method simply returns the value of self.x from the object itself. 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 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. It will perform a more complex task.

1 class Point:

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

3

​4 def \_\_init\_\_(self, initX, initY):

5 """ Create a new point at the given coordinates. """

6 self.x = initX

7 self.y = initY

8

​

9 def getX(self):

10 return self.x

11

​12 def getY(self):

13 return self.y

14

​15 def distanceFromOrigin(self):

16 return ((self.x \*\* 2) + (self.y \*\* 2)) \*\* 0.5

17

​18

​19 p = Point(7, 6)

20 print(p.distanceFromOrigin())

21

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

SavvyPython

View:

Textbook

Social

Search

User

Scratch Activecode

Help

This Chapter

17.7. Objects as Arguments and Parameters

You can pass an object as an argument in the usual way. We’ve already seen this in some of the turtle examples where we passed the turtle to some function like drawRectangle so that the function could control and use whatever turtle instance we passed to it.

Here is a simple function called distance involving our new Point objects. The job of this function is to figure out the distance between two points.

1 import math

2

​3 class Point:

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

5

​6 def \_\_init\_\_(self, initX, initY):

7 """ Create a new point at the given coordinates. """

8 self.x = initX

9 self.y = initY

10

​11 def getX(self):

12 return self.x

13

​14 def getY(self):

15 return self.y

16

​17 def distanceFromOrigin(self):

18 return ((self.x \*\* 2) + (self.y \*\* 2)) \*\* 0.5

19

​20 def distance(point1, point2):

21 xdiff = point2.getX() - point1.getX()

22 ydiff = point2.getY() - point1.getY()

23

​24 dist = math.sqrt(xdiff\*\*2 + ydiff\*\*2)

distance takes two points and returns the distance between them. Note that distance is not a method of the Point class. You can see this by looking at the indentation pattern. It is not inside the class definition. The other way we can know that distance is not a method of Point is that self is not included as a formal parameter. In addition, we do not invoke distance using the dot notation.

17.8. Converting an Object to a String

When we’re working with classes and objects, it is often necessary to print an object (that is to print the state of an object). Consider the example below.

1

class Point:

2

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

3

​

4

def \_\_init\_\_(self, initX, initY):

5

""" Create a new point at the given coordinates. """

6

self.x = initX

7

self.y = initY

8

​

9

def getX(self):

10

return self.x

11

​

12

def getY(self):

13

return self.y

14

​

15

def distanceFromOrigin(self):

16

return ((self.x \*\* 2) + (self.y \*\* 2)) \*\* 0.5

17

​

18

​

19

p = Point(7, 6)

20

print(p)

21

​

Activity: 17.8.1 ActiveCode (chp13\_classesstr1)

The print function shown above produces a string representation of the Point p. The default functionality provided by Python tells you that p is an object of type Point. However, it does not tell you anything about the specific state of the point.

We can improve on this representation if we include a special method call \_\_str\_\_. Notice that this method uses the same naming convention as the constructor, that is two underscores before and after the name. It is common that Python uses this naming technique for special methods.

The \_\_str\_\_ method is responsible for returning a string representation as defined by the class creator. In other words, you as the programmer, get to choose what a Point should look like when it gets printed. In this case, we have decided that the string representation will include the values of x and y as well as some identifying text. It is required that the \_\_str\_\_ method create and return a string.

1 class Point:

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

3

​4 def \_\_init\_\_(self, initX, initY)

5 """ Create a new point at the given coordinates. """

6 self.x = initX

7 self.y = initY

8

​9 def getX(self):

10 return self.x

11

​12 def getY(self):

13 return self.y

14

​15 def distanceFromOrigin(self):

16 return ((self.x \*\* 2) + (self.y \*\* 2)) \*\* 0.5

17

​18 def \_\_str\_\_(self):

19 return "x=" + str(self.x) + ", y=" + str(self.y)

20

​21 p = Point(7, 6)

22 print(p)

23

When we run the program above you can see that the print function now shows the string that we chose.

Now, you ask, don’t we already have an str type converter that can turn our object into a string? Yes we do!

And doesn’t print automatically use this when printing things? Yes again!

But, as we saw earlier, these automatic mechanisms do not do exactly what we want. Python provides many default implementations for methods that we as programmers will probably want to change. When a programmer changes the meaning of a special method we say that we override the method. Note also that the str type converter function uses whatever \_\_str\_\_ method we provide.

17.9. Instances as Return Values

Functions and methods can return objects. This is actually nothing new since everything in Python is an object and we have been returning values for quite some time. The difference here is that we want to have the method create an object using the constructor and then return it as the value of the method.

Suppose you have a point object and wish to find the midpoint halfway between it and some other target point. We would like to write a method, call it halfway that takes another Point as a parameter and returns the Point that is halfway between the point and the target.

1 class Point:

2

​

3 def \_\_init\_\_(self, initX, initY):

4 """ Create a new point at the given coordinates. """

5 self.x = initX

6 self.y = initY

7

​8 def getX(self):

9 return self.x

10

​11 getY(self):

12 return self.y

13

​14 def distanceFromOrigin(self):

15 r eturn ((self.x \*\* 2) + (self.y \*\* 2)) \*\* 0.5

16

​17 def \_\_str\_\_(self):

18 return "x=" + str(self.x) + ", y=" + str(self.y)

19

​20 def halfway(self, target):

21 mx = (self.x + target.x) / 2

22 my = (self.y + target.y) / 2

23 r return Point(mx, my)

24

The resulting Point, mid, has an x value of 4 and a y value of 8. We can also use any other methods since mid is a Point object.

In the definition of the method halfway see how the requirement to always use dot notation with attributes disambiguates the meaning of the attributes x and y: We can always see whether the coordinates of Point self or target are being referred to.

**Chapter 17 Glossary**

class

A class can be thought of as a template for the objects that are instances of it. It defines a data type. A class can be provided by the Python system or be user-defined.

constructor

Every class has a “factory”, called by the same name as the class, for making new instances. If the class has an initializer method, this method is used to get the attributes (i.e. the state) of the new object properly set up.

initializer method

A special method in Python (called \_\_init\_\_) that is invoked automatically to set a newly-created object’s attributes to their initial (factory-default) state.

instance

An object whose type is of some class. Instance and object are used interchangeably.

instantiate

To create an instance of a class, and to run its initializer.

method

A function that is defined inside a class definition and is invoked on instances of that class.

object

A compound form of data that is often used to model a thing or concept in the real world. It bundles together the data and the operations that are relevant for that thing or concept. It has the type of its defining class. Instance and object are used interchangeably.

object-oriented programming

A powerful style of programming in which data and the operations that manipulate it are organized into classes and methods.

object-oriented language

A language that provides features, such as user-defined classes and inheritance, that facilitate object-oriented programming.

**Chapter 17 Excercises**

Add a distanceFromPoint method that works similar to distanceFromOrigin except that it takes a Point as a parameter and computes the distance between that point and self.

Add a method reflect\_x to Point which returns a new Point, one which is the reflection of the point about the x-axis. For example, Point(3, 5).reflect\_x() is (3, -5)

Add a method slope\_from\_origin which returns the slope of the line joining the origin to the point. For example,

>>> Point(4, 10).slope\_from\_origin()

2.5

What cases will cause your method to fail? Return None when it happens.

The equation of a straight line is “y = ax + b”, (or perhaps “y = mx + c”). The coefficients a and b completely describe the line. Write a method in the Point class so that if a point instance is given another point, it will compute the equation of the straight line joining the two points. It must return the two coefficients as a tuple of two values. For example,

>>> print(Point(4, 11).get\_line\_to(Point(6, 15)))

>>> (2, 3)

This tells us that the equation of the line joining the two points is “y = 2x + 3”. When will your method fail?

Add a method called move that will take two parameters, call them dx and dy. The method will cause the point to move in the x and y direction the number of units given. (Hint: you will change the values of the state of the point)

Given three points that fall on the circumference of a circle, find the center and radius of the circle.