20.7. 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.

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):
```

```
<__main__.Point object>
```

Activity: 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.

Whatever string the `__str__` method for a class returns, that is the string that will print when you put any instance of that class in a print statement. For that reason, the string that a class's `__str__` method returns should usually include values of instance variables. If a point has `x` value 3 and `y` value 4, but another point has `x` value 5 and `y` value 9, those two Point objects should probably look different when you print them, right?

Take a look at the code below.

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):
```

```
x = 7, y = 6
```

Activity: 2 -- ActiveCode (chp13_classesstr2)

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 a `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!

However, 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 method we say that we **override** the method. Note also that the `str` type converter function uses whatever `__str__` method we provide.

Check Your Understanding

1. Create a class called Cereal that accepts three inputs: 2 strings and 1 integer, and assigns them to 3 instance variables in the constructor: `name`, `brand`, and `fiber`. When an instance of `Cereal` is printed, the user should see the following: "[name] cereal is produced by [brand] and has [fiber integer] grams of fiber in every serving!" To the variable name `c1`, assign an instance of `Cereal` whose name is `"Corn Flakes"`, brand is `"Kellogg's"`, and fiber is `2`. To the variable name `c2`, assign an instance of `Cereal` whose name is `"Honey Nut Cheerios"`, brand is `"General Mills"`,

and fiber is `3`. Practice printing both!

Save & Run 5/15/2021, 12:25:18 PM - 9 of 9 Show in CodeLens

```
class Cereal:
    def __init__(self,name,brand,fiber):
        self.name = name
        self.brand = brand
        self.fiber = fiber

    def __str__(self):
        return "{} cereal is produced by {} and has {} grams of fiber in every serv
c1=Cereal("Corn Flakes","Kellogg's",2)
c2=Cereal("Honey Nut Cheerios","General Mills",3)
print(c1)
print(c2)
```

```
Corn Flakes cereal is produced by Kellogg's and has 2 grams of fiber in every serving!
Honey Nut Cheerios cereal is produced by General Mills and has 3 grams of fiber in every serving!
```

Activity: 3 -- ActiveCode (ac_ch13_classstr_01)

Result	Actual Value	Expected Value	Notes	
Pass	'Corn ...ving!'	'Corn ...ving!'	Testing that c1 prints correctly.	Expand Differences
Pass	'Honey...ving!'	'Honey...ving!'	Testing that c2 prints correctly.	Expand Differences

You passed: 100.0% of the tests

You have attempted 4 of 3 activities on this page

20.6. Objects as Arguments and Parameters">
20.8. Instances as Return Values">
bjects as Arguments and Parameters">
✔ Completed. Well Done!
20.8. Instances as Return Values">Next Section - 20.8. Instances as Return Values

| Back to top