Classes and Instances

Introduction

In this lesson, you'll take a look at class and instance objects in Python and how to create them. A Python class can be thought of as the blueprint for creating a code object (or **instance object**). It has both the layout for new objects as well as the ability to create those objects. When you **initialize** or make a new instance object from a class, you are essentially pressing a button on an assembly line that instantly rolls out a new instance object. For example, if you were dealing with a `Car` class, you would get a brand new car from the assembly line. In cases where you want to create multiple objects, you can see how this functionality would be extremely useful.

Objectives

You will be able to:

- Describe a class and how it can be used to create objects
- Describe an instance object
- Create an instance of a class

Defining a class

Imagine you are starting a ride share business. Let's call it *fuber*. All rides generally have the same basic information. They have a driver, passenger(s), origin, destination, car information, and price. You plan on having a pretty large client base, so, you could imagine having many rides being taken every day.

So, you will need to have a way to bundle up and operate on all the above information about a particular ride. And as the business would take off, you are going to need to create rides over and over.

How can you use Python to help make this process easier? Using Python classes. You can create a `Ride` class that can produce ride instance objects, which would bundle all the information and common operations for a particular ride.

As mentioned earlier, a class can be thought of as the blueprint that defines how to build an instance object. The `Ride` class is different from an actual ride instance object just as the blueprint for a house is different from the actual house.

Here is what our `Ride` class would look like in Python:

```
class Ride:
   # code for distance attribute
   # code for time attribute
   # code for price attribute
   # code to start a ride
   # code to end a ride
```

You can see that you use the keyword `class` to define a Python class. Similar to functions, all the code in a class should be indented. To end the class, you simply stop indenting.

> **Note:** Python's convention is that all classes should follow the UpperCaseCamelCase convention. That is the class should begin with a capital letter and all other words in the name should also be capitalized. This is otherwise referred to as CamelCase.

It's not enough to simply declare a class in Python. All classes need to have a block of code inside them or else you will get an error. Let's see this below:

In [1]:
```
class Ride:
```

```
  File "<ipython-input-1-81f6a00d9a7f>", line 2

    ^
SyntaxError: unexpected EOF while parsing
```

So, let's add a block of code to our `Ride` class and see what happens. Python has a keyword `pass` which you can use in this instance to tell our code to do nothing and continue executing. `pass` can be used for times where a block of code is syntactically necessary, like defining a class or function. Feel free to read more about `pass` [here (https://docs.python.org/2/tutorial/controlflow.html#pass-statements)](https://docs.python.org/2/tutorial/controlflow.html#pass-statements).

In [7]:
```
class Ride_test:
    pass
```

Woo! No error. So, you have now successfully defined our `Ride` class. Let's try to create an `instance` of this class. Again, you can think of these instances as objects of the `Ride` class that contain information about a single ride.

In [8]:
```
first_ride = Ride_test()
print(first_ride)
```
```
<__main__.Ride_test object at 0x7fe550155c10>
```

Okay, you ***instantiated*** our first ride! You did this by invoking, or calling the `Ride()` class. You invoke a class the same way you do with functions, by adding parentheses `()` to the end of the class name, (i.e. `Ride()`).

Instantiate means to bring a new object to life (off the assembly line). You instantiated a new ride when you invoked the class, `Ride()` , which made a new ride in our rideshare program.

Each individual object produced from a class is known as an **instance** or **instance object**. Our variable, `first_ride` , points to an `instance` of the ride class. You can be sure it is an instance of the `Ride` class by looking at the object you created.

When you printed `first_ride` above, you saw that it says it is a `Ride object` . This tells us not only which class it comes from, the `Ride` class, but also that it is an instance since it says `object` .

You can see this distinction more clearly by printing both the class and an instance of that class:

In [9]:
```
print(Ride_test)
print(first_ride)
```
```
<class '__main__.Ride_test'>
<__main__.Ride_test object at 0x7fe550155c10>
```

Great, now let's dive a little deeper into instances. You made one already, let's make a couple more and compare them:

In [10]:
```
second_ride = Ride_test()
third_ride = Ride_test()
print(first_ride)
print(second_ride)
print(third_ride)
```
```
<__main__.Ride_test object at 0x7fe550155c10>
<__main__.Ride_test object at 0x7fe550155be0>
<__main__.Ride_test object at 0x7fe5501550d0>
```

Three rides! Alright, let's look at these. They seem pretty much the same, except the funny numbers at the end. Those are the IDs that represent a place in memory where the computer stores these objects. Additionally, since the IDs are unique, this means that each instance object is a **completely unique object** although they are all created from the same `Ride` class. You can prove this by comparing the objects below:

In [11]:
```
print(first_ride is second_ride)
print(first_ride == second_ride)
print(first_ride is first_ride)
```
```
False
False
True
```

As you can see, `first_ride` is only equal to itself even though at this point these objects all have identical attributes and methods (or lack thereof) with the exception of their IDs in our computer's memory.

Summary

In this lesson, you learned about what you use classes for and how to define them. They are the blueprints for creating instance objects and they allow us to create instance objects with the same or similar attributes and methods. However, all instance objects are produced with unique IDs, making them unique objects.