4.9.3 Object Oriented Programming

Maria has been wondering about something: in expressions like `df.head()`, what exactly does the period refer to? To answer her question, she'll need to understand the concept of object-oriented programming.

Object-oriented programming (OOP) is a way of organizing information in programming. It is based on the concepts of classes and objects. Imagine a mold for a toy car at a toy factory. The mold can be used to create identical toy cars, but each car is a separate **object**. If a child places a sticker on one particular car, for example, only that car will have the sticker. In object-oriented programming, the **class** is like the mold, and the **object** is like a particular toy car.

Let's take a look at an example of a class and objects in Python. To create a class called `Cat`, let's try the following code.

```
class Cat:
    def __init__(self, name):
        self.name = name
```

In this example, the keyword `class` is used to create a class. Then, the `__init__` method is a special method called a "class constructor" that Python calls every time a new instance of the class is created. This method takes two arguments: `self` and `name`. The `self` argument refers to a specific `Cat` object that is created when this method is called. The second argument, `name`, will be specified each time a new Cat object is created. Let's look at how to do this.

```
first_cat = Cat('Felix')
print(first_cat.name)
```

Here, an object whose `name` is `Felix` is created, or instantiated. Since it is assigned to the variable `first_cat`, `first_cat.name` returns `Felix`. Pay attention to the dot notation in this example. The `name` attribute does not belong to all `Cat` objects; it belongs to the **specific** `Cat` object called `first_cat`.

SKILL DRILL

Create another object of the `Cat` class. Its name should be `Garfield`, and the object should be assigned to the variable `second_cat`. What does `second_cat.name` return?

In OOP, an object can have attributes, such as `name` in the example above. It can also perform actions through functions. Let's now create a `Dog` class.

```
class Dog:
    def __init__(self, name, color, sound):
        self.name = name
        self.color = color
        self.sound = sound
```

```
    def bark(self):
        return self.sound + ' ' + self.sound
```

This class contains three attributes: `name`, `color`, and `sound`. It also contains a method, or a function that is associated with an object. This means that an object created from this `Dog` class can now perform an action through the `bark` method. Let's instantiate an object.

```
first_dog = Dog('Fido', 'brown', 'woof!')
print(    first_dog.name)
print(first_dog.color)
first_dog.bark()
```

The `first_dog`'s `bark` method returns the `first_dog`'s `sound` property twice: because the `sound` property is `woof!` for this object, calling the `bark` method returns `woof! woof!`.

SKILL DRILL

Instantiate another object of the `Dog` class. Its name should be `Lady`, its color should be `blonde`, and its sound should be `arf!`. The object should be assigned to the variable `second_dog`. Call this object's `bark` method.

In a nutshell, object-oriented programming uses classes and objects to organize information: an object can hold attributes, and it can also run functions. When a function is associated with an object, it is called a method. If an object's attributes are like its adjectives and nouns ("color" and "name" for example), then its methods are like its verbs ("bark").

Finally, remember that if you wanted to preview a Pandas DataFrame called `df`, you would call its `head` method.

```
df.head()
```

The dot notation used here should be a clue that the Pandas uses object-oriented programming to create an object called `df`, which represents a DataFrame, and that this object has a method called `head`, which returns the first five rows of data. You will see many more examples of such objects, their attributes, and their methods in the rest of the course.