Full Stack Development
Sesi 5

PYTHON:
OOP IN PYTHON

Object-oriented programming (OOP)

Object-oriented programming is a programming paradigm that provides a means of structuring programs so that properties and behaviors are bundled into individual objects.

For instance, an object could represent a person with properties like a name, age, and address and behaviors such as walking, talking, breathing, and running. Or it could represent an email with properties like a recipient list, subject, and body and behaviors like adding attachments and sending.

Class Introduction

For example, let's say you want to track employees in an organization. You need to store some basic information about each employee, such as their name, age, position, and the year they started working.

One way to do this is to represent each employee as a list:

```
raka = ["Raka Ardhi", 28, "CurDev", 2265]
spock = ["Spock", 35, "Science Officer", 2254]
mccoy = ["Leonard McCoy", "Chief Medical Officer", 2266]
```

There are a number of issues with this approach.

First, it can make larger code files more difficult to manage. If you reference raka[0] several lines away from where the raka list is declared, will you remember that the element with index 0 is the employee's name?

Second, it can introduce errors if not every employee has the same number of elements in the list. In the mccoy list above, the age is missing, so mccoy[1] will return "Chief Medical Officer" instead of Dr. McCoy's age.

A great way to make this type of code more manageable and more maintainable is to use classes.

Classes vs Instances

Classes are used to create user-defined data structures. Classes define functions called methods, which identify the behaviors and actions that an object created from the class can perform with its data.

A class is a blueprint for how something should be defined. It doesn't actually contain any data. The Dog class specifies that a name and an age are necessary for defining a dog, but it doesn't contain the name or age of any specific dog.

While the class is the blueprint, an instance is an object that is built from a class and contains real data. An instance of the Dog class is not a blueprint anymore. It's an actual dog with a name, like Miles, who's four years old.

Put another way, a class is like a form or questionnaire. An instance is like a form that has been filled out with information. Just like many people can fill out the same form with their own unique information, many instances can be created from a single class.

Define a Class

All class definitions start with the class keyword, which is followed by the name of the class and a colon. Any code that is indented below the class definition is considered part of the class's body.

The properties that all Dog objects must have are defined in a method called .__init__(). Every time a new Dog object is created, .__init__() sets the initial state of the object by assigning the values of the object's properties. That is, .__init__() initializes each new instance of the class.

You can give .__init__() any number of parameters, but the first parameter will always be a variable called self. When a new class instance is created, the instance is automatically passed to the self parameter in .__init__() so that new attributes can be defined on the object.

Define a Class

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

Notice that the .__init__() method's signature is indented four spaces. The body of the method is indented by eight spaces. This indentation is vitally important. It tells Python that the .__init__() method belongs to the Dog class.

In the body of .__init__(), there are two statements using the self variable:

self.name = name creates an attribute called name and assigns to it the value of the name parameter.
self.age = age creates an attribute called age and assigns to it the value of the age parameter.
Attributes created in .__init__() are called instance attributes. An instance attribute's value is specific to a particular instance of the class. All Dog objects have a name and an age, but the values for the name and age attributes will vary depending on the Dog instance.

Define a Class

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

Notice that the .__init__() method's signature is indented four spaces. The body of the method is indented by eight spaces. This indentation is vitally important. It tells Python that the .__init__() method belongs to the Dog class.

In the body of .__init__(), there are two statements using the self variable:

self.name = name creates an attribute called name and assigns to it the value of the name parameter.
self.age = age creates an attribute called age and assigns to it the value of the age parameter.
Attributes created in .__init__() are called instance attributes. An instance attribute's value is specific to a particular instance of the class. All Dog objects have a name and an age, but the values for the name and age attributes will vary depending on the Dog instance.

Class and Instance Attributes

Now create a new Dog class with a class attribute called .species and two instance attributes called .name and .age:

```
>>> class Dog:
...     species = "Canis familiaris"
...     def __init__(self, name, age):
...         self.name = name
...         self.age = age
```

To pass arguments to the name and age parameters, put values into the parentheses after the class name:

```
>>> buddy = Dog("Buddy", 9)
>>> miles = Dog("Miles", 4)
```

After you create the Dog instances, you can access their instance attributes using dot notation:

```
>>> buddy.name
'Buddy'
>>> buddy.age
9
```

Instance Methods

```
class Dog:
    species = "Canis familiaris"

    def __init__(self, name, age):
        self.name = name
        self.age = age

    # Instance method
    def description(self):
        return f"{self.name} is {self.age} years old"

    # Another instance method
    def speak(self, sound):
        return f"{self.name} says {sound}"
```

This Dog class has two instance methods:

- .description() returns a string displaying the name and age of the dog.
- .speak() has one parameter called sound and returns a string containing the dog's name and the sound the dog makes.

Instance Methods

```
>>> miles = Dog("Miles", 4)

>>> miles.description()
'Miles is 4 years old'

>>> miles.speak("Woof Woof")
'Miles says Woof Woof'

>>> miles.speak("Bow Wow")
'Miles says Bow Wow'
```

External References

OOP - Visit Here