## **What Is Object-Oriented Programming (OOP)?**

Object-oriented Programming, or *OOP* for short, is a [programming paradigm](http://en.wikipedia.org/wiki/Programming_paradigm) which 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 a name property, age, address, etc., with behaviors like walking, talking, breathing, and running. Or an email with properties like recipient list, subject, body, etc., and behaviors like adding attachments and sending.

Put another way, object-oriented programming is an approach for modeling concrete, real-world things like cars as well as relations between things like companies and employees, students and teachers, etc. OOP models real-world entities as software objects, which have some data associated with them and can perform certain functions.

Another common programming paradigm is *procedural programming* which structures a program like a recipe in that it provides a set of steps, in the form of functions and code blocks, which flow sequentially in order to complete a task.

The key takeaway is that objects are at the center of the object-oriented programming paradigm, not only representing the data, as in procedural programming, but in the overall structure of the program as well.

## **Classes in Python**

Focusing first on the data, each thing or object is an instance of some *class*.

The primitive data structures available in Python, like numbers, strings, and lists are designed to represent simple things like the cost of something, the name of a poem, and your favorite colors, respectively.

What if you wanted to represent something much more complicated?

For example, let’s say you wanted to track a number of different animals. If you used a list, the first element could be the animal’s name while the second element could represent its age.

How would you know which element is supposed to be which? What if you had 100 different animals? Are you certain each animal has both a name and an age, and so forth? What if you wanted to add other properties to these animals? This lacks organization, and it’s the exact need for *classes*.

Classes are used to create new user-defined data structures that contain arbitrary information about something. In the case of an animal, we could create an Animal() class to track properties about the Animal like the name and age.

It’s important to note that a class just provides structure—it’s a blueprint for how something should be defined, but it doesn’t actually provide any real content itself. The Animal() class may specify that the name and age are necessary for defining an animal, but it will not actually state what a specific animal’s name or age is.

It may help to think of a class as an *idea* for how something should be defined.

## **Python Objects (Instances)**

While the class is the blueprint, an *instance* is a copy of the class with *actual* values, literally an object belonging to a specific class. It’s not an idea anymore; it’s an actual animal, like a dog named Roger who’s eight years old.

Put another way, a class is like a form or questionnaire. It defines the needed information. After you fill out the form, your specific copy is an instance of the class; it contains actual information relevant to you.