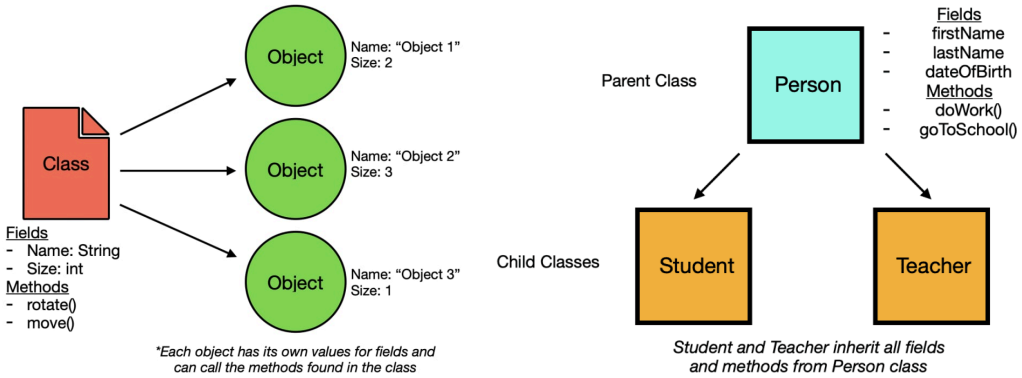
What Is Object-Oriented Programming?



Object-oriented programming (OOP) is a programming paradigm that organizes data and functions into units called objects. These objects are often instances of classes. I just threw a lot of new terms at you, so let's clarify them.

A **programming paradigm** is a strategy or style of programming. While some languages are geared towards a specific paradigm, others (like Python) support multiple paradigms. OOP is a paradigm which Python supports particularly well. Other paradigms include functional programming and procedural programming, which we won't cover in depth here, but I encourage you to research them if you're interested in different programming styles.

In OOP, an **object** is a fundamental building block that bundles together data (attributes) and behaviors (methods). It's a general term that refers to any entity created from a class.

Classes are templates used to create objects. They define the structure for an object, like a blueprint. A class consists of **attributes** (data members, or properties; things we can say about the object) and **methods** (functions; things the object can do).

An object that is created from a class is called an **instance** of that class. Let's look at an example to help these concepts gel.

Imagine you're building a racing game. You might want to create a class named "Car", which serves as a template for creating all the cars in the game. Each car in the game would be an instance of the Car class.

The Car class defines data members like "color" and "top_speed", and methods like "accelerate" and "brake". These data members and methods outline the characteristics and behaviors of a car within the game's context.

Each instance of the Car class might have its own unique attributes. You can create a red car object with a top speed of 160 mph, and a separate blue car object with a top speed of 145 mph. Classes enable programmers to create and manage distinct objects that behave independently while sharing the same underlying structure. This is an idea I want to really emphasize. OOP is such a powerful programming style because it allows us to write something one time and reuse it elsewhere. It supports the classic programming rule of thumb: **DRY (Don't Repeat Yourself)**.

OOP is built on four important principles: **encapsulation**, **inheritance**, **polymorphism**, and **abstraction**. Let's talk about those.

Encapsulation

Encapsulation involves bundling related data and methods into a single box, which we call an object. This box helps keep the information safe by not letting anything outside of the box directly touch it. This way, we prevent accidental or unauthorized changes. In Python, this is more of a suggestion than a strictly enforced rule. We usually mark information that should stay inside the box with one underscore (_) for "protected" attributes, and two underscores (__) for "private" attributes. Other languages like Java, C#, and C++ actually enforce encapsulation more strictly.

Inheritance

Inheritance allows us to create new classes (**child** classes) from existing ones (**parent** classes). The parent, or base class, acts as a blueprint for the child, derived class. This allows us to reuse relevant parts of the old class, which eliminates redundant code.

Polymorphism

Polymorphism allows us to use one single interface to control many different things. In some ways it's like a universal remote control; the same buttons do similar things on different devices. We can use operators and methods on different classes without specifying exactly what they are. In Python, we call this "**duck typing**" (if it looks like a duck and quacks like a duck, we assume it must be a duck). In other words, we focus more on *what an object can do*, rather than *what an object is*.

Two completely different objects might both have a method with the same name, and both would be allowed to use that method without us having to specify which object it is. For example, imagine a Shark class and a Dolphin class that both have a method called "swim". We can call the swim method on an instance of either class without specifying whether it's a Shark or a Dolphin; as long as the object can swim, we can call the method (even if both methods don't work exactly the same way).

Abstraction

Abstraction is a technique for making complicated systems easier to handle. It does this by showing only what's necessary to the user and hiding the complicated parts under the hood. Python doesn't enforce this as strictly as some other languages, but it does offer a tool called the **abc** (abstract base class) module. This tool lets you create classes with "abstract" methods, meaning if a class uses these methods, it must define how these methods work.