A Brief Introduction

This lesson welcomes you to the world of Object Oriented Programming.

We'll cover the following

Procedural Programming

If you’re here, you’re probably familiar with the basics of programming already and have used *methods* in your programs at some point.

Procedural programming is one programming paradigm among many.

In procedural programming, a program is divided into smaller parts called methods. These **methods** are the **basic entities** used to construct a program. One of the main advantages of procedural programming is code reusability. However, the implementation of a complex real-world scenario becomes a difficult task unwieldy.

Enter: Object-Oriented Programming

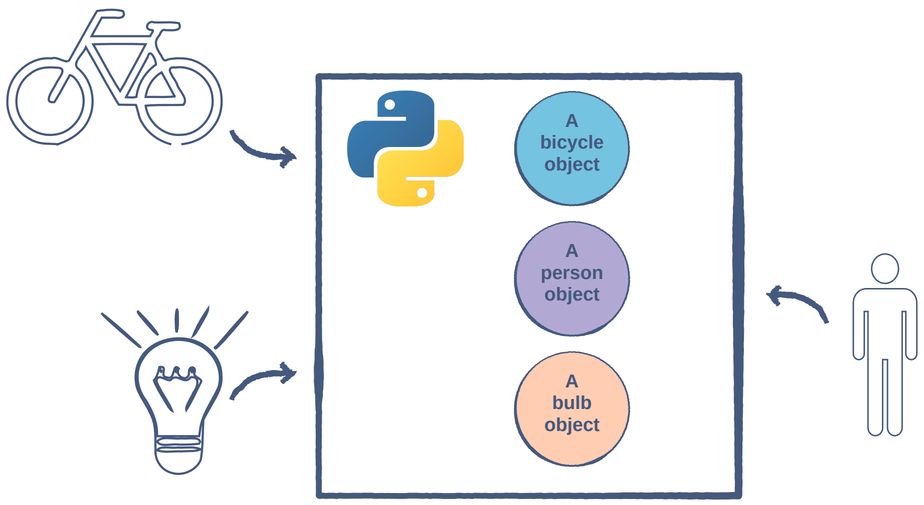
Object-oriented programming, also referred to as **OOP**, is a programming paradigm that includes, or relies, on the concept of classes and objects.

The basic entities in object-oriented programming are **classes and objects.**

Programming isn’t much use if you can’t model real-world scenarios using code, right? This is where Object-Oriented Programming comes into play.

The basic idea of OOP is to divide a sophisticated program into a bunch of **objects** talking to each other.

Objects in a program frequently represent real-world objects.



Many other objects serve application logic and have no direct, real-world parallels. They manage things like authentication, templating, request handling, or any of the other myriad features needed for a practical application.

Anatomy of Objects and Classes

Objects may contain data in the form of *fields* (variables) and methods to operate on that data.

Think about the real-world objects around you. *What are the characteristics of these objects*? Take the example of a *light bulb*. It has a **state**, i.e., either it is *on* or *off*. It also has a **behavior**, i.e., when you turn it on it lights up, and when turned off, it stops spreading light. To conclude this, one can say:

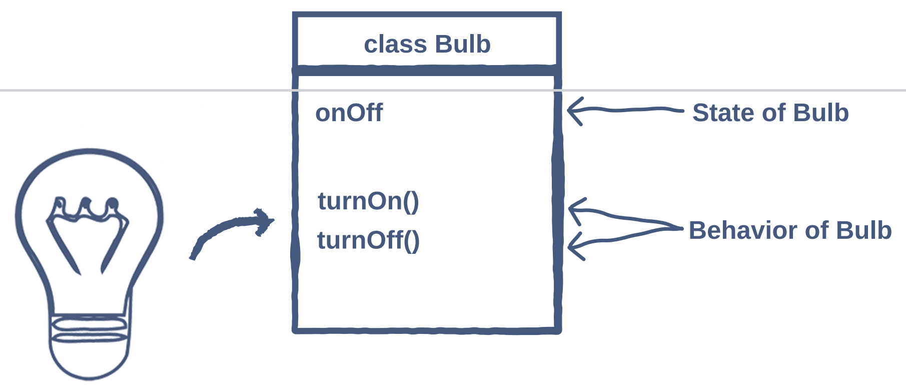
> Objects are a collection of **data** and their **behaviors**.

Interesting! Isn't it? But the question is *"where do these objects come from?"*

Well, the answer to the above question is **classes**.

> A **Class** can be thought of as a *blueprint* for creating objects.

The below illustration shows what a `LightBulb` class should look like:



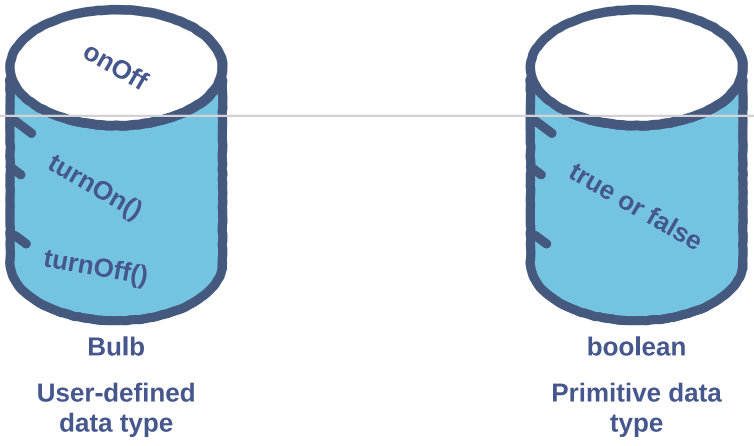
1 of 3

From the above illustration, you can see that the state of the object is generally modeled using *variables* in a class, and the behavior is modeled using *methods*.

There can be many different objects of the same class. Each can be in an independent state, but they all share the same behavior and characteristics.

User-Defined Data Types

It can be inferred from the discussion above that classes are user-defined data types implemented using primitive data types, e.g. `boolean`, `int`, `char` etc. While primitive data types only focus on modeling the state of the object, **user-defined data types** can encapsulate state and its behaviors into a unit.



In the next lesson, we'll discuss some different Object-Oriented Programming languages, Python amongst them.

Next →

Introduction to Objects and Classes

✓ Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/a-brief-introduction__introduction-to-object-oriented-programming__learn-object-oriented-programming-in-python)