A Brief Introduction

This lesson welcomes you to the world of object-oriented programming.

WE'LL COVER THE FOLLOWING

Procedural Programming

Presumably, you are familiar with the basics of programming and have used methods, also referred to as functions, in your programs.

Procedural programming is a programming paradigm. In procedural programming, a program is divided into smaller parts called methods.

The **basic entities** in procedural programming are **methods**.

The focal point of the procedural programming technique is to use methods for code reusability. This means that once a method is implemented in a program, it can be called on any number of times to perform the certain operation(s) rather than repetitively writing code for these operations. However, using this paradigm to implement a complex real-world scenario can be a difficult task.

What is Object-Oriented Programming?

Programming is used to solve real-world problems and it won't make much sense if one can't model real-world scenarios using programming languages, right? This is where object-oriented programming comes into play!

> *Object-oriented programming*, also referred to as **OOP**, is a programming

> paradigm that relies on the concept of classes and objects.

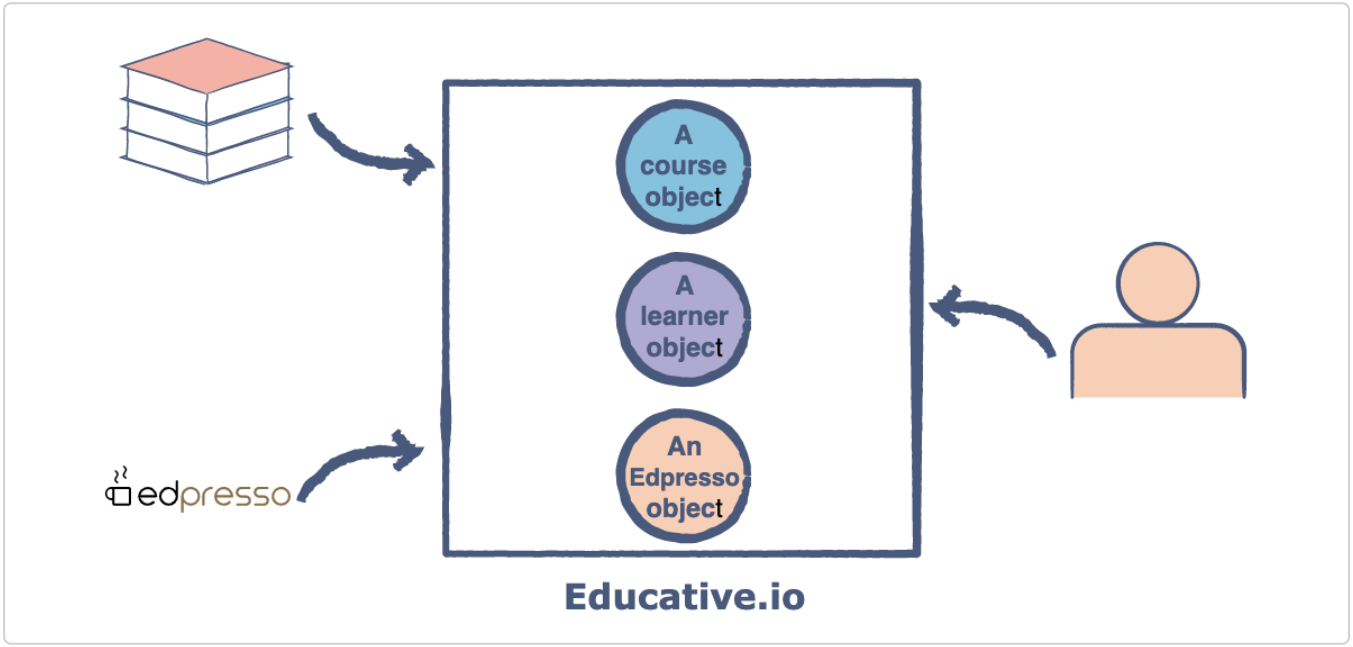
In the real world, when we look around us, there are objects everywhere. These objects interact with each other and that is how the real world works. Furthermore, we find that many objects are similar and consider them to be instances of the same class. For example, Dog is a class whereas your pet dog is an instance of that class. Your pet has certain commonalities with all other dogs. That's why we consider them to be instances of the same class. Composing computer programs as collections of objects (instead of methods) is the base of object-oriented programming. The simulation of this real-world scenario in programming explores the concepts at the base of object-oriented programming.

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

Objects in a program/application frequently represent real-world objects.

Let's suppose you are about to design the educative.io platform. A very basic model will consist of the following objects:

- Courses
- Learners
- Edpresso Shots



In software applications, there are many objects that serve the application logic and have no direct real-world parallel objects. Some of the most common tasks performed by these objects are managing authentication, templating, request handling, or any of the other myriad features needed for a working 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 a *calculator* for example. It has a **state**, i.e., either it is *on* or *off*. It also has **behaviors**, i.e., you can perform addition, subtraction, multiplication, division and many other operations on numbers. This means we can say:

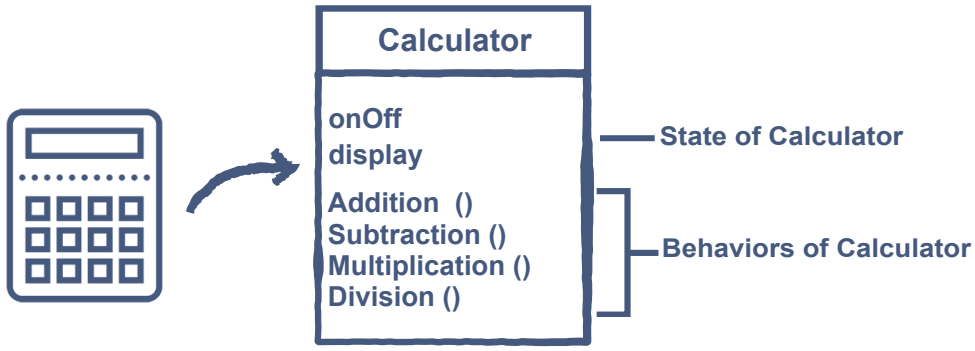
> Objects have **state(s)** and **behavior(s)**.

Interesting, isn't it? But the question is *"where do 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 illustration below shows what a `Calculator` class should look like.



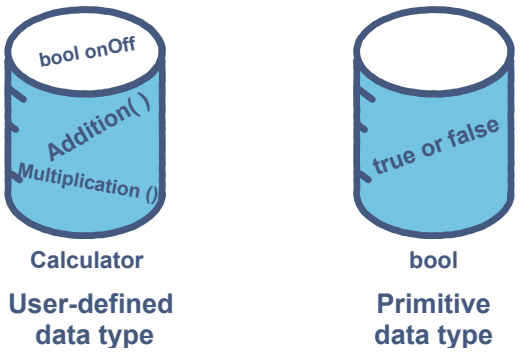
If you were to implement a calculator object in a computer program, variables could represent it's state and functions could represent its behavior.

Just as a real-world object's state and behavior can't be separated, a class in object-oriented programming encapsulates behavior and state in a single unit.

> One can create multiple objects of the same class. Each object can be in an independent state, but they all share the same behavior and characteristics.

C# provides us many primitive data types to store different types of data. The variables of these primitive types can only store values. Talking in a more OOP specific way we can say that these variables focus only on modeling the state of the object.

OOP based languages enable programmers to create powerful customized variables using classes. Unlike primitive data types, theses user-defined data types, also known as classes, can model the state as well as the behavior of an object. As a result, when a variable of this user-defined data type is created, it can do way more than merely storing a single value.



It can be inferred from the discussion above that classes are user-defined data types implemented using the primitive data types e.g. `bool`, `int`, `char` etc.

We will leave this discussion for now and come back to it in a later section.

In the next lesson, you will get to know about modern Object-Oriented languages and how C# is one of them.