Object-Oriented Programming[1]

[1] aka OOP

Robin Dawes

November 14, 2021

Turns "traditional" programming inside out by making objects (or more precisely, *models* of objects) the fundemental building blocks of programs.

In the Beginning ...

In the early days of computing, most people focused on developing and improving the algorithms used to solve problems. Data was viewed as what you fed into the algorithm. If the question of how the data should be organized came up, it was in the context of trying to make the algorithms run faster.

The Dawn of OOP

In the 1960's, a new way of looking at programming started to appear. The fundamental idea was that programming could be viewed as the process of creating simplified versions of "things" (either real or conceptual) and then defining operations on those things, and interactions between things. People had already realized that data often occurs as groups of values all related to a particular thing. But in the new way of thinking, called Object-Oriented Programming, the ability to create clear and self-contained models became central - even primary.

For example, a real estate agent's data might consist of groups of data, each group containing a property address, asking price, number of bedrooms, etc.

The expression "Object-Oriented Programming" was coined by Alan Kay around 1967.

Carrying forward the real estate agent scenario, an object-oriented program designed to manage her inventory might start by defining a generic informational model of a house. The model might include information such as the address, the square-footage of the property and building, an image of the house, the present owner's name, the price, and so on. The program would define this class (or type) of object, and then create instances of the class, each one populated with values for the various information fields. Each instance would represent one property that the agent is trying to sell. Each instance of the house model would be a distinct entity within the program. There would be specific functions designed to access and modify

the individual pieces of information about a house - for example, to change the asking price - and other functions defined to operate on a collection of houses - for example, to find the house with the lowest asking price.

Now suppose the real estate agent decides to branch out and deal with business properties as well as residential properties. The model for business properties would have some pieces of information in common with the house model - for example, address, price, and square-footage - but there would be other pieces of information that would only be relevant to one type of property - for example, a business property might not have a "number of bedrooms" field in its description.

The OOP approach would be to create a more generic model called "Property" which would specify the pieces of information and operations that the two types of property have in common, and then define the "Residential" and "Business" models as extensions of the "Property" model, each specifying the additional pieces of information that are particular to them.

Going even further, the program could define a generic model for clients of the real estate agent. This "Client" model could then be extended by a "Buyer" model and a "Seller" model. Functions could then be defined that would permit interactions between clients and properties. For example, a function could be written to make a list of potential buyers for a particular property, or to schedule a showing of a property to a buyer. The scheduled visit to the property could be stored in another type of object - probably called an "Event" or something similar. The entire program design process revolves around defining the models of the objects we need, and defining the functions that manipulate and relate them.

In Python, these models are created using the `class` keyword. Each class definition usually contains an `__init__` function that identifies the pieces of information that are relevant to an instance of the class, and other functions that relate to the instance as well. The class can also contain variables and functions that are not specific to a particular instance. The instance variables and functions are distinguished from the class variables and functions by the keyword `self`.

Much more complete information about these Python-specific details will be found in the posted demo code samples.

Fundamental Concepts of OOP

OBJECT-ORIENTED PROGRAMMING has developed into a huge branch of computer science, and it involves many advanced subtopics that you will encounter in future courses. For now, I'm just going to list four foundational ideas of OOP .

Encapsulation

Encapsulation refers to the idea that all the information and functionality that relates to an object should be contained within the object ... and so far as is possible, it should be hidden inside the object. By this I mean that the information should only be accessible through a defined set of functions provided in the class itself. This aspect of OOP is often referred to as INFORMATION HIDING

I think of it as "NOYB" information management ... as in "None Of Your Business" - but I doubt that anyone else uses this term!

Abstraction

When we make a model of a type of object, we have to decide which pieces of information are required - sometimes there are many pieces of information that we decide to ignore. The process of deciding what information to include and what to leave out is known as ABSTRACTION.

For a house, we might decide to leave out the colour of the window-frames.

Inheritance

When one class extends another class, it inherits the whole template of its parent class - all of the variables and functions. This is useful because if there is a particular variable or function that we need in many of our classes, we can avoid having to define the variable or function in each of the classes. Instead, we can put the definition in a generic class and have all of our important classes extend (and inherit from) the generic class.

Polymorphism

Polymorphism refers to the fact that different classes can have variables and/or functions that have the same name, but completely different definitions. For example, if our classes represent different types of animal, then the "find_food()" function in the Tiger class would

probably be quite different from the "find_food()" function in the Sheep class.