Computer Programming Methodologies

Sequential Programming

In sequential programming, lines of code are written and executed one after the other ("First do this, then do this, then do this, etc..."). The disadvantage to this method is that you sometimes have to repeat the same code over and over again.

Procedural Programming

In procedural programming, lines of code are grouped together into a bundle called a procedure (or function) to perform a certain task. When you want to perform that task, you simply direct the program to run the code in the function (called 'calling' the function).

The strength of writing code like this is that it is much more efficient. You don't have to write the same code over and over again. You write it once in a function and then just call the function when you need it. Procedural programming also allows you write neater code that is easier to read.

Object-Oriented Programming

With object-oriented programming **(OOP),** you create separate files called **classes** that define not only the data type of data but also the types of operations (functions) that can be applied to the data.

The idea behind OOP is that all these classes can interact together to create a working program. In procedural programming, the basic building block is the function, but in OOP, the building block is the class (data and function). Your program creates an **object** from one of the classes. Custom classes can be written from scratch to produce specialized types of content such as a car in a racing program.

In other words: An object is a collection of data (variables) and methods (functions) that act on those data. A class is a blueprint for the object.

Python, we define a class using the keyword *class*.

```
class MyClass:
```

The procedure to create an object in Python looks like this:

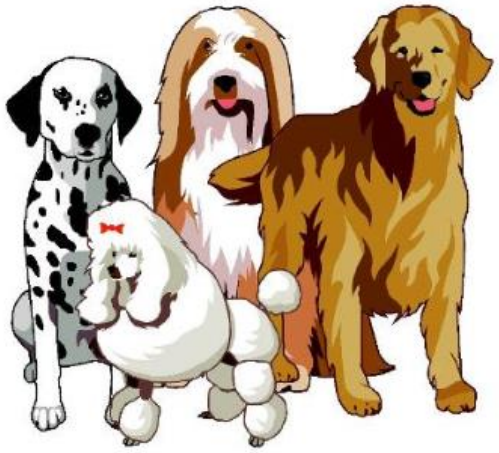
```
ob = MyClass()
```

(https://www.programiz.com/python-programming/class)

Advantages of OOP

There are many advantages to OOP. First, all of the code is no longer in one big file. The classes are separate files that can be worked on by different people at the same time. Secondly, these classes are also generally reusable which means that the more you program, the more classes you have to work with. When you start a new project, you are never starting from scratch because you have a library of classes upon which to draw.

Objects



To understand the concept of objects, it often helps to think of a real-world object such as a dog. A dog could be said to have properties such as name, colour, breed, age, and behaviours such as barking, eating, running.

OOP objects also have properties and behaviours. Using object-oriented techniques, you can model a real-world object.

Instances

Continuing with the real-world analogy of a dog, consider that there are dogs with different names, colours, breeds, and ages and with different ways of eating and behaving. However, regardless of their individual differences, all dogs are members of a general type or class—the class of dogs.

A class is a definition of a type, while an object is an instance of a class. In other words, the instance is the actual object that can be manipulated. The difference between classes and objects is critical. You cannot pet the class (the concept dog) but you can pet the object (the instance) of a dog.

The act of creating an object is called instantiation. Programmers talk about instantiating an object. This means that they create an object of a given type as defined by the class definition.

In OOP, a class defines a blueprint for a type of object. The characteristics and behaviours that belong to a class are jointly referred to as members of that class. The characteristics (in the dog example, the name, age, breed, and colour) are called properties of the class and are represented as variables; the behaviours (barking, eating, and running) are called methods of the class, and these are represented as functions.

Inheritance

One of the primary benefits of OOP is that you can **extend** a class to create a subclass. Writing subclasses lets you reuse code. Instead of recreating all the code common to both classes, you can simply extend an existing class.

When you extend a class, the subclass **inherits** all the properties and methods of the original class. The subclass usually defines additional methods and properties or **overrides** methods or properties defined in the superclass.

For example, you could build a superclass called Animal, which contains common characteristics and behaviours of all animals. Next, you could build several subclasses that inherit from the Animal superclass and add characteristics and behaviours specific to that type of animal.

Polymorphism

The word ‘poly' means many and the word ‘morph' means form. The idea behind polymorphism is that an object can take many forms. Using polymorphism, classes can override methods of their superclasses and define specialized implementations of those methods.

For example, you might start with a class called Mammal that has play() and sleep() methods. You then create Monkey and Dog subclasses to extend the Mammal class. The subclasses override the play() method from the Mammal class to reflect the habits of those particular kinds of animals. Monkey implements the play() method to swing from trees and Dog implements the play() method to fetch a ball. Because the sleep() functionality is similar among the animals, you would use the superclass implementation.

Example: A Banking Program

Suppose a bank has created a program to manage customers' accounts.

An Account class is created to allow the following functions to be performed.

- Deposit money into the account.
- Withdraw money from the account.
- Find out how much money is in the account.

These three functions are part of the Account class public interface. Clients use them but they do not know the details of how they work. The details of how they work are part of the private implementation of the class.

An ATM (Automatic Teller Machine) could be a client of the Account class. For example, when a customer at the ATM presses the button to request an account balance, the ATM asks the Account class for the current balance.

How the Account class arrives at the current balance (e.g., querying a database, performing calculations etc.) is not of concern to the ATM; the ATM simply expects a value to return so that it can provide the balance to the customer.

Encapsulation

The word encapsulate means to encompass or contain. In programming, encapsulation is the process of combining elements to create a new entity. OOP languages rely on encapsulation to create classes. A well planned class should fully encapsulate all the aspects of its area of responsibility. Encapsulation endorses the idea of "do one thing and do it well".

For example, the Account class discussed above should encapsulate all the information related to using customer accounts (obtaining balances, getting money out, and putting money in). It would be a poor design to have these functions split up between several different classes or to have functions included in the Account class that have nothing to do with customer accounts.

Example: A Fast Food Restaurant

Next time you go through the drive-through window at your favourite fast-food restaurant, think about how the restaurant performs encapsulation.

Steps that occur at the drive-through:

- A customer rolls down his car window, talks into a speaker, and places the order.
- The order is acknowledged and the customer drives forward to pay.
- An employee leans out of a window and takes the money.
- At the same time, somewhere in the background, another employee is preparing the food.
- The customer drives forward to another window and picks up the food—hot and ready to eat!

The restaurant has encapsulated four responsibilities:

1. taking orders
2. taking money
3. preparing food
4. delivering food

Two important characteristics of encapsulation:

1. Each responsibility should be fully implemented.

Suppose the procedure for taking money was not fully implemented by the restaurant. Think of how inconvenient it would be if instead of using cash or your bank card to pay for your food order at the restaurant, you had to go to the bank and arrange for a transfer for $6.88 into the restaurant's bank account.

2. The customer should NOT know the implementation details.

When a customer places an order for food, he knows that there is a lot going on inside the restaurant (people are cooking, others are packaging, others are cleaning) but he really doesn't care about these details; he just wants the food!

There are two additional concepts that work with encapsulation. They are data hiding and delegation.

Data Hiding

Data hiding means that the internal state including the data and implementation of a class is hidden from its clients.

Delegation

Delegation is the opposite of encapsulation. One principle of encapsulation is that it restricts itself to a single area of responsibility. If there are tasks outside its area of responsibility, a class should delegate those responsibilities.

For example, the fast food restaurant we discussed does not grow its own food or manufacture its own paper goods. These responsibilities are delegated to other companies so that it can focus on what it does best, taking your money and giving you food.

Degrees of Object Orientation

- Pure OO languages treat everything as an object. Examples: Python, Ruby, Scala, Smalltalk, Eiffel, Emerald, JADE
- Languages designed mainly for OO programming, but with some procedural elements include: Java, C++, C#, Delphi/Object Pascal, VB.NET.
- Languages that are historically procedural languages, but have some OO features include: PHP, Perl, Visual Basic, MATLAB.
- Languages with supports which may be used to resemble OO programming, but without all features of object-orientation include: JavaScript, Lua,

(wikipedia)