# OOP Concepts in Python

## Objects

As already discussed in the previous tutorial, an object is a physical entity, whereas class is a logical defintion. An object can be anything like - a student while designing a school's record registry, or a pen in stationary's item management program, or a car in manufacture's car database program. There are three important characteristics by which it is identified, they are:

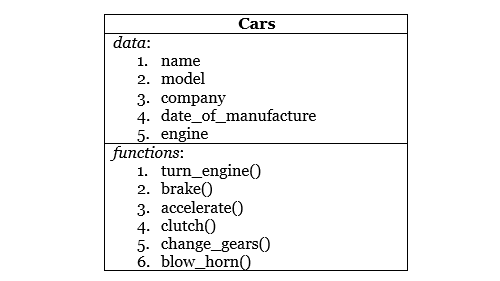
1. **Identity**: Identity refers to some piece of information that can be used to identify the object of a class. It can be the name of the student, the company name of the car, etc.
2. **Properties**: The attributes of that object are called properties. Like age, gender, DOB for a student; or type of engine, number of gears for a car.
3. **Behaviour**: Behaviour of any object is equivalent to the functions that it can perform. In OOP it is possible to assign some functions to objects of a class. Taking the example forward, like a student can read/write, the car can accelerate and so on.

## Class

A class is a blueprint where attributes and behaviour is defined. For example, if Mahatma Gandhi, Sachin Tendulkar, you and me are objects, then **Human Being** is a class. An object is the fundamental concept of OOP but classes provide an ability to define similar type of objects.

In class, both data, and the functions that will be operating on that data are bundled as a unit.

For example, let's say there is a class named Car, which has some basic data like - name, model, brand, date of manufacture, engine etc, and some functions like **turn the engine on**, **apply brakes**, **accelerate**, **change gear**, **blow horn** etc. Now that all the basic features are defined in our class Car, we can create its objects by setting values for properties name, model etc, and the object of the class Car will be able to use the functions defined in it.



Behaviour

State

## Data Hiding

Data hiding helps us to define the privacy of data from the outside world or to be precise, from other classes. The reason behind doing this is to create several levels of accessing an object's data and prevent it from accidental modification. Also, hiding, or setting up privacy levels, can be done for functions as well.

In OOP, data inside the classes can be defined as **public**, **private** or **protected**. Private data or function are the ones that cannot be accessed or seen from outside the class whereas, public data or functions can be accessed from anywhere. Protected data or functions, more or less act like public but should not be accessed from outside.

## Abstraction of Data

Classes use the concept of abstraction. A class encapsulates the relevant data and functions that operate on data by hiding the complex implementation details from the user. The user needs to focus on what a class does rather than how it does.

## Encapsulation

Encapsulation, is one of the core reason for the existence of an object. Since the beginning we have been talking about objects, its data, functions, privacy etc., now it's time to know how is all this kept bounded. The answer is encapsulation. Much as it may sound like a capsule, it is pretty much the same. Here we try to encapsulate the data and functions together which belongs to the same class.

## Inheritance

As explained in the previous tutorial, inheritance is about defining a set of core properties and functions in one place and then re-using them by inheriting the class in which they are defined.

Python supports **Simple**, **Multiple** and **MultiLevel** Inheritance. We will be covering these in details in inheritance tutorial.

## Polymorphism

[Polymorphism](https://www.studytonight.com/python/python-polymorphism), or **Poly + Morph**, means **"many forms**. Precisely, Polymorphism is the property of any function or operator that can behave differently depending upon the input that they are fed with.

Polymorphism can be achieved in tow different forms, they are:

### Function Overloading

In OOP, it is possible to make a function act differently using function overloading. All we have to do is, create different functions with same name having different parameters. For example, consider a function add(), which adds all its parameters and returns the result. In python we will define it as,

def add(a, b):

return a + b

This is capable of executing function calls like:

>>> add(4,5)

The above add function will always take 2 numbers as input, but what if you want to add 3 numbers at once or maybe 4 numbers.

Hence, in OOP you can simply define the function add once again, this time with 3 parameters, and this mechanism is known as **Function Overloading**.

# to add 3 numbers

def add(a, b, c):

return a + b + c

# to add 4 numbers

def add(a, b, c, d):

return a + b + c + d

Now we can call add() function with two, three or four parameters. As you can see here, function add() now has multiple forms.

### [Operator Overloading](https://www.studytonight.com/python/python-operator-overloading)

You know what operators are: **Addition**, **Division**, **Multiplication** etc. Python is capable of reading operators differently depending upon the situation. For example,

>>>2 + 5

It will give you output 7, but doing

>>> "hello" + "world"

Gives you output "helloworld". We can see here, how + operator when used with numbers performs mathematical addition operation but when used with strings it performs concatenation.

Similarly, the multiplication operator also acts differently based on the datatype of the variables with which it is used. For example,

>>> 3\*7

Gives, 21, while on string,

>>> "hello"\*3

it gives, "hellohellohello". This is yet another example of operator overloading.