OOP Terminology

Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.

Inheritance: The transfer of the characteristics of a class to other classes that are derived from it.

Instance: An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.

Instantiation: The creation of an instance of a class.

Method: A special kind of function that is defined in a class definition.

Object: A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.

Recursion In Python:

Recursion is a way of programming or coding a problem, in which a function calls itself one or more times in its body. Usually, it is returning the return value of this function call. If a function definition fulfils the condition of recursion, we call this function a recursive function.

Recursive Functions in Python

Example

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

Sets in Python:

A set contains an unordered collection of unique and immutable objects. The set data type is, as the name implies, a Python implementation of the sets as they are known from mathematics. This explains, why sets unlike lists or tuples can't have multiple occurrences of the same element.

Operator overloading:
The assignment of more than one function to a particular operator.Suppose you have created a Vector class to represent two-dimensional vectors, what happens when you use the plus operator to add them? Most likely Python will yell at you.

You could, however, define the __add__ method in your class to perform vector addition and then the plus operator would behave as per expectation −

Example

```
class Vector:
   def __init__(self, a, b):
      self.a = a
      self.b = b

   def __str__(self):
      return 'Vector (%d, %d)' % (self.a, self.b)

   def __add__(self,other):
      return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print v1 + v2
```

When the above code is executed, it produces the following result −

```
Vector(7,8)
```