OOP Terminology

- **Class-** A user-defined prototype for an object that defines a set of attributes that characterize any object of the class.The attribues are data memebers (class variables and instance variables) and methods, accessed via dot notation.
- **Class variable-** A variable that is shared by all instancs of a class. Class variables are defined within a class but outside any of the class's methods.
- **Instance variable-** A variable that is defined inside a method and belongs only to the current instances of a class.
- **Function overloading-** The assisgnment of more than one behavior to a particular function. The operation performed varies by the types of objects or arguments involved.
- **Operator overloading-** The assignment of more than one function to a particular orperator.

Creating Class

```
The class statement creates a new class definition. The class has a documentation string, which can be accessed via ClassName.__doc__.
```

In [14]:
```
class Employee:
    'Common base class for all employees'
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    def displayCount():
        print("Total Employee %d"%Employee.empCount)

    def displayEmployee(self):
        print("Name : ",self.name,", Salary: ",self.salary)
```

In [15]:
```
Employee.__doc__
```

Out[15]:
```
'Common base class for all employees'
```

Creating instance of object

In [16]:
```
emp1 = Employee("Zara", 2000)
emp2 = Employee("Manni", 5000)

emp1.displayEmployee()
```

```
Name :  Zara , Salary:  2000
```

In [17]:
```
Employee.displayCount()
```

```
Total Employee 2
```

add, remove or modify attribues

In [23]:
```
emp1.age = 7
```

In [24]:
```
emp1.age
```

Out[24]:
```
7
```

In [25]:
```
emp1.age =8
```

In [26]:
```
emp1.age
```

Out[26]:
```
8
```

In [27]:
```
del emp1.age
```

In [28]:
```
emp1.age
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-28-0d569e33fad3> in <module>
----> 1 emp1.age

AttributeError: 'Employee' object has no attribute 'age'
```

Instead of using the normal statements to access attributes, you can use the following functions

- **getattr(obj,name[,default])** - to access the attribues of object
- **hasattr(obj,name)** - to check if an attribute exits or not.
- **setattr(obj,name,value)** - to set an attribute. If attribue does not exist then it would be created.
- **delattr(obj,name)** - to delete an attribute.

```
In [29]: hasattr(emp1,'age')
```

```
Out[29]: False
```

```
In [31]: getattr(emp1,'salary')
```

```
Out[31]: 2000
```

```
In [32]: setattr(emp1,'age',8)
```

```
In [33]: emp1.age
```

```
Out[33]: 8
```

```
In [34]: delattr(emp1,'age')
```

```
In [35]: hasattr(emp1,'age')
```

```
Out[35]: False
```

Built-in class attributes

```
__dict__ - dictionary containing class's namespace.
__doc__ - Class documentation string
__name__ - class name
__module__ - Module name in which the class is defined. This attribue is "__main__" in interactive
mode.
__bases__ - A possibly empty tuple containing the base classes, in the order of their occurance in
the base class list
```

In [36]:
```
Employee.__doc__
```

Out[36]:
```
'Common base class for all employees'
```

In [37]:
```
Employee.__name__
```

Out[37]:
```
'Employee'
```

In [38]:
```
Employee.__module__
```

Out[38]:
```
'__main__'
```

In [39]:
```
Employee.__bases__
```

Out[39]:
```
(object,)
```

In [40]:
```
Employee.__dict__
```

Out[40]:
```
mappingproxy({'__module__': '__main__',
              '__doc__': 'Common base class for all employees',
              'empCount': 2,
              '__init__': <function __main__.Employee.__init__(self, name, salary)>,
              'displayCount': <function __main__.Employee.displayCount()>,
              'displayEmployee': <function __main__.Employee.displayEmployee(self)>,
              '__dict__': <attribute '__dict__' of 'Employee' objects>,
              '__weakref__': <attribute '__weakref__' of 'Employee' objects>})
```

Inheretence

Insted of starting from scratch, you can create a class by deriving it from a preexisting class by listing the parent class in parentheses after the new class name.

The clild class inherits the attributes of its parent class, and you can use those attributes as if theywere defined in the child class. A child class can also override data members and methods from the parent.

In [41]:
```
class Parent:
    parentAttr = 100
    def __init__(self):
        print("Calling parent constructor")

    def parentMethod(self):
        print("Calling parent method")

    def setAttr(self, attr):
        Parent.parentAttr = attr

    def getAttr(self):
        print("Parent attribute : ",Parent.parentAttr)
```

In [42]:
```
class Child(Parent):
    def __init__(self):
        print("Calling child constructor")

    def childMethod(self):
        print("Calling child method")
```

In [43]:
```
c = Child()
```

```
Calling child constructor
```

In [44]:
```
c.childMethod()
```

```
Calling child method
```

In [45]:
```
c.parentMethod()
```

```
Calling parent method
```

In [46]:
```
c.setAttr(200)
```

In [47]:
```
c.getAttr()
```

```
Parent attribute :  200
```

Similar way, you can drive a class from multiple parent classes

```
class C(A,B) # subclass of A and B
```

In [48]:
```
issubclass(Child,Parent)
```

Out[48]:
```
True
```

In [49]:
```
isinstance(c,Child)
```

Out[49]:
```
True
```

Overriding Methods

You can always ovverride your parent class methods. One reason for overriding parent's methods is because you may wnat special or different functionality in your subclass.

In [50]:
```
class Parent:
    def myMethod(self):
        print("Calling parent method")

class Child(Parent):
    def myMethod(self):
        print("Calling child method")
```

In [51]:
```
c = Child()
```

In [52]:
```
c.myMethod()
```

```
Calling child method
```

Base overloading methods

```
Following table lists some generic functionality that you can override in your own class

__init__(self[,args...]) # Constructor with any optional arguments
__del__(self) # Destructor
__repr__(self) # Evaluates string representation
```

```
__str__(self) # Printable string representation
__cmp__(self) # Object comparison
```

Overloading Operators

Suppose you have created a vector class to represent two-dimensional vectors, what happens when you use the plus operator to add them ? Python will yell at you.

You could, however, define the __add__ method in your class to perform vector addition and then pllus operator would behave as expected.

In [79]:
```
class Vector:
    def __init__(self, a, b):
        self.a = a
        self.b = b

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

    def __repr__(self):
        return "Vector is ({}, {})".format(self.a,self.b)

    def __str__(self):
        return "Vector ({}, {})".format(self.a,self.b)
```

In [80]:
```
v1 = Vector(2,10)
v2 = Vector(5,-2)
print(v1+v2)
```

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

If we dont define __repr__ method

In [81]:
```
v1
```

Out[81]:
```
Vector is (2, 10)
```

In [82]:
```
print(v1)
```

```
Vector (2, 10)
```

```
If __repr__ defined, it can get called automatically when we print the value of an instance of a
class for which we define this method. __str__ method is used for similar but not identical
purpose, that may get precedence if we have also defined it.
```

Data Hiding

An object's attributes may or may not be visible outside the class definition. You need to name attributes with a double underscore prefix, and those attribues then are not be directly visible to outsiders.

In [83]:
```
class JustCounter:
    __secretCount = 0

    def count(self):
        self.__secretCount += 1
        print(self.__secretCount)
```

In [84]:
```
counter = JustCounter()
counter.count()
counter.count()
```

```
1
2
```

In [85]:
```
print(counter.__secretCount)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-85-7aced26a4181> in <module>
----> 1 print(counter.__secretCount)

AttributeError: 'JustCounter' object has no attribute '__secretCount'
```

Python protects those members by internally changing the name to include the class name. You can access such attributes as object._className__attrName.

In [86]:
```
print(counter._JustCounter__secretCount)
```

```
2
```