封装的意义在于保护或者防止代码被被外部在无意中破坏, 在面向对象程序设计中一个类被看作是一个中心的原素并且和使用它的对象结合的很密切，所以保护它不被其它的函数意外的修改就显得格外重要， 通过封装可以实现要访问该类的代码和数据必须要通过严格的接口控制，

例子:

例： class student:

score = 0

s = student()

s.score = 1000 在绑定属性时，如果我们直接把属性暴露出去，虽然写起来很简单，但是，没办法检查参数，导致可以把成绩随便改：

```
s = Student()

s.score = 9999
```

这显然不合逻辑.

我们需要在类中把一个变量定义的属性定义为私有 Python中的表现形式为__foo，通过在类中定义getter和setter接口来获取和设置该变量的值

private # 权限限制最小的访问修饰符 称之为私有的 被其修饰的属性及方法 只能被类本身访问 连子类也不能访问 Python中的表现形式为__foo

封装的优点：

1. 良好的封装能够减少耦合。
2. 类内部的结构可以自由修改。
3. 可以对成员变量进行更精确的控制。

4. 隐藏信息，实现细节。

```
private 关键字在python的表达形式
定义属性 __name
定义方法 __getname(self)

#-*-coding:utf8-*-
class c1:
    def __init__(self, name, age):
        self.__name = name
        self.age = age

    def __getName(self):
        return self.__name
    def getAge(self):
        return self.age
x = c1('liushuo', 18)
print x.age
#print x.__name   ##报错  因为 private 的属性不可以被实例调用
print x.getAge()
#print x.__getName() ##报错 因为private的方法不可以被实例调用

class c2(c1):
    def __init__(self, name, age, job):
        c1.__init__(self, name, age)
        self.job = job
    def getJob(self):
        return self.job
    def changName(self, value):
        self.__name = value
        return self.__name
y = c2('zhangsan', 19, 'python')
print y.getJob()
print y.changName('devops')
#print y.__name #报错
```

##双下划线开头的 代表的是私有类型的变量或方法 这个只能被类本身 以及子类 所访问和修改 但是不能被实例修改

为了限制score的范围，可以通过一个set_score()方法来设置成绩，再通过一个get_score()来获取成绩，
这样，在set_score()方法里，就可以检查参数：

```
class Student(object):


    def get_score(self):
         return self._score

    def set_score(self, value):
        if not isinstance(value, int):
            raise ValueError('score must be an integer!')
        if value < 0 or value > 100:
            raise ValueError('score must between 0 ~ 100!')
        self._score = value
```

现在，对任意的Student实例进行操作，就不能随心所欲地设置score了：

```
>>> s = Student()

>>> s.set_score(60) # ok!>>> s.get_score()
60>>> s.set_score(9999)
Traceback (most recent call last):
  ...
ValueError: score must between 0 ~ 100!
```

但是，上面的调用方法又略显复杂，没有直接用属性这么直接简单。
有没有既能检查参数，又可以用类似属性这样简单的方式来访问类的变量呢？对于追求完美的Python程序员来说，这是必须要做到

的!

Python中的写法

```
class Student(object):

@property
def score(self):
    return self._score

@score.setter
def score(self, value):
    if not isinstance(value, int):
        raise ValueError('score must be an integer!')
    if value < 0 or value > 100:
        raise ValueError('score must between 0 ~ 100!')
    self._score = value
```