面向对象(二)

类属性property

作用：

- property可以定义一个方法名为私有属性的名字， 让用户可以访问， 但不能修改， 保护数据的安全性;
- @属性名.setter在给属性赋值时， 先做判断；
- @属性名.deleter使用内置del删除属性时，自动执行的内容;

In [19]:

```
class Student(object):
    def __init__(self, name, age, score):
        self.name = name
        self.__age = age
        self.__score = score

    @property
    def age(self):
        return  self.__age


    @age.setter
    def age(self, value):
        if isinstance(value, int) and 0 < value <= 150:
            self.__age = value
            print "age update ok."
        else:
            self.__age = 0
            print "invalid age value"

    @age.deleter
    def age(self):
        print "age: %s, can not delete" %(self.__age)




class MathStudent(Student):
    pass



student1 = Student("fentiao", 10, 100)
print student1.age
student1.age = 180
print student1.age
del student1.age


# student1.age = 100
```

```
10
invalid age value
0
age: 0, can not delete
```

- 第2种方式: property(fget, fset, fdel, fdoc)

In [1]:

```
class Student(object):
    def __init__(self, name, age, score):
        self.name = name
        self.__age = age
        self.__score = score


    def get_age(self):
        return  self.__age

    def set_age(self, value):
        if isinstance(value, int) and 0 < value <= 150:
            self.__age = value
            print "age update ok."
        else:
            self.__age = 0
            print "invalid age value"

    def del_age(self):
        print "age: %s, can not delete" %(self.__age)


    age = property(get_age, set_age, del_age, "age sttribute")

class MathStudent(Student):
    pass



student1 = Student("fentiao", 10, 100)
print student1.age
student1.age = 180
print student1.age
del student1.age
```

```
10
invalid age value
0
age: 0, can not delete
```

pproperty应用: 信息分页显示

- 主机信息有很多， 为了美观， 分页显示；
- 当用户选择第n页， 显示该页需要的数据从哪条开始， 哪条结束;
- 将start， end返回给后端， 将需要的数据响应给前端;

In [ ]:

```
hostinfo = ['172.25.254.'+str(i)  for i in range(1,101)]

# print hostinfo
class Pager(object):
    def __init__(self, current_page, per_items=10):
        # 显示第x页的数据；
        self.__current_page = current_page
        # 每一页显示多少条数据
        self.__per_items = per_items

    @property
    def start(self):
        val = (self.__current_page-1)*self.__per_items
        return val


    @property
    def end(self):
        val = self.__current_page * self.__per_items
        return  val

p = Pager(3, 5)
# print type(p.start())
# print type(p.end())
print hostinfo[p.start:p.end]


# current_pag=1, per_items=10,   start=0, (1-1)*10  end=10
# current_pag=2, per_items=10,   start=10,(2-1)*10   end=20
# print  hostinfo[0:10]
# print  hostinfo[10:20]
```

私有成员和共有成员

- 私有属性/方法: 类内部可以访问, 对象不能访问； 子类不能访问， 子类的对象不能访问;
- 共有属性/方法: 私有属性可以访问和不可以访问的， 都可以访问;

In [9]:

```
class Student(object):
    def __init__(self, name, score):
        self.name = name
        # 在类里面， 双下划线开头的属性，只在类里面生效， 外部调用不生效;
        # python解释器将self.__属性名间接转换为self._类名__属性名
        self.__score = score


    def  __judge(self):
        if 90 <= self.__score <=100:
            print "A"
        elif 80 <= self.__score <90:
            print "B"
        else:
            print "C"



class MathStudent(Student):
    def getscore(self):
        print self.__score

student1 = Student("fentiao", 100)

print student1.name
# 私有属性， 不能访问;
# print student1.__score
print student1._Student__score

# 私有方法
student1._Student__judge()


student2 = MathStudent("fendai", 80)
print student2.name
# 私有属性
# print student2.__score

# 不能访问
# student2.getscore()
```

```
fentiao
100
A
fendai
```

改变类的字符串显示 __str__和 __repr__

- __str__ : 当打印对象时自动调用;
- __repr__ : 当在交互式环境中， 直接输入对象时， 自动调用;
- 如果 __str__ 没有定义时， 那么打印对象自动调用 __repr__ ；

- 功能: 简化脚本测试和调试时的实例输出;

In [20]:

```
class Student(object):
    def __init__(self, name, sid):
        self.__name = name
        self.__sid = sid


    def __str__(self):
        return "Student(%s)"  %(self.__name)


    # 交互式环境中使用；
    def __repr__(self):
        return "<%s %s>" %(self.__name, self.__sid)

if __name__ == "__main__":
    s = Student("westos", 001)
    print s
    s
```

```
<__main__.Student object at 0x2150510>
```

自定义字符串的格式

- __format__

In [29]:

```
_formats = {
    'ymd' : '{d.year}-{d.month}-{d.day}',
    'mdy' : '{d.month}/{d.day}/{d.year}',
    'dmy' : '{d.day}/{d.month}/{d.year}'
}


class Date(object):
    def __init__(self, year, month, day):
        self.year = year
        self.month = month
        self.day = day


    def __format__(self, code):
        if code == '':
            code = 'ymd'
        fmt = _formats[code]
        return fmt.format(d=self)




d = Date(2018, 1, 9)
print format(d)
print format(d, 'dmy')
print format(d, 'mdy')
```

```
2018-1-9
9/1/2018
1/9/2018
```

索引:__getitem__和__setitem__和__delitem__

In [50]:

```
class Student(object):


    country = "china"

    def __init__(self, name, age):
        self.name = name
        self.__age = age


    def get_age(self):
        return self.__age


    def __getitem__(slef, item):
        """

        当访问某个属性时， 自动执行的魔术方法;
        """
        return None
    def __setitem__(self, key, value):
        """
        当给某个属性赋值时， 自动调用的方法;
        """
        self.__dict__[key] = value
        print "setitem ok"

    def __delitem__(self, key):
        """
        当删除某个属性时， 自动执行的方法;

        """
        del self.__dict__[key]
        print "delete ok"



s = Student("westos", 10)

print s.__dict__


s['name']
name = s['name']
s['name'] = 'westos'
s['name']
del s['name']
```

```
{'name': 'westos', '_Student__age': 10}
```

```
{'name': 'westos', '_Student__age': 10}
setitem ok
delete ok
```

切片: __getslice和__setslice__和__delslice__

In [56]:

```
class Student(object):


    country = "china"

    def __init__(self, name, age):
        self.name = name
        self.__age = age


    def __getslice__(self, i, j):
        print "__setslice__", i, j


    def __setslice__(self, i, j, item):
        print "__getslice__", i, j, item

    def __delslice__(self, i, j):
        print "__delslice__", i, j


s = Student("westos", 12)


s[0:3]

s[0:3] = [12, 23, 45]

del s[0:3]
```

```
__setslice__ 0 3
__getslice__ 0 3 [12, 23, 45]
__delslice__ 0 3
```

类支持比较操作(<, <=, ==, !=, >, >=)

In [4]:

```
class Room(object):
    def __init__(self, name, length, width):
        self.name = name
        self.length = length
        self.width = width
        self.square = self.length * self.width



class House(object):
    def __init__(self, name, style):
        self.name = name
        self.style = style
        self.rooms = list()


    def __str__(self):
        return "%s %s %s" %(self.name, self.style, self.rooms)

    @property
    def all_square(self):
        return sum([i.square for i in self.rooms])

    def __eq__(self, other):
        return self.all_square == other.all_square


    __nq__ = lambda self, other: self.all_square !=  other.all_square

    def __lt__(self, other):
        return self.all_square < other.all_square

    def __le__():
        pass

    def __gt__():
        pass

    def __ge__():
        pass

    def add_room(self, room):
        self.rooms.append(room)



h1 = House('house1', "style1")
h2 = House('house2', "style2")
room1 = Room("room1", 1, 1)
room2 = Room("room2", 2, 2)
room3 = Room("room3", 3, 3)
```

```
room3 = Room('room3', 3, 3)


h1.add_room(room1)
h1.add_room(room2)


h2.add_room(room2)
h2.add_room(room3)



h1 == h2

h1 < h2


h1 != h2
```

Out[4]:

True

迭代__iter__

In [14]:

```
class Student(object):
    def __init__(self, name):
        self.name = name
        self.scores = [100, 90, 89]


    def add_score(self, score):
        self.scores.append(score)


    def __iter__(self):
        # 生成一个迭代对象
        return iter(self.scores)



s = Student("fentiao")
s.add_score(97)

print s.scores


from collections import Iterable
print isinstance(s, Iterable)


print "按照学生成绩迭代显示: ",
for i in s:
    print i,
```

```
[100, 90, 89, 97]
True
按照学生成绩迭代显示:  100 90 89 97
```

元类('type')

- 在python中，一切皆对象; Linux下一切皆文件;

In [21]:

```
# Student是个类， 实质上是个对象
# Student类是type类的一个实例化;
class Student(object):
    print "student class start........"

s = Student()

print type(s)
print type(Student)
```

```
student class start........
<class '__main__.Student'>
<type 'type'>
```

动态的创建类

In [24]:

```
def choose(name):
    if name == "chinese":
        class ChineseStudent(object):
            pass
        return ChineseStudent

    else:
        class MathStudent(object):
            pass
        return MathStudent



myclass = choose('chinese')
print myclass
```

```
<class '__main__.ChineseStudent'>
```

type动态创建类

- type(对象)
- type(类名, 元组方式存储父类，属性)

In [25]:

```
type(1)
```

Out[25]:

```
int
```

In [26]:

```
# class Student(object):
#     def add_score(self):
#         print "add score......"


def add_score(self):
    print "add score......."

Student = type("Student", (object, ), {"add_score": add_score})


s = Student()
s.add_score()
```

```
add score.......
```

with语句(语句上下文管理)

In [31]:

```
class File(object):
    def __init__(self, file_name, file_mode='r'):
        # 文件对象
        self.file_obj = open(file_name, file_mode)

    def __enter__(self):
        return self.file_obj

    def __exit__(self, exc_type, exc_val, exc_tb):
        return self.file_obj.close()


with File('/etc/passwd') as f:
    pass


# with open("/etc/passwd") as f:
#     f.read()
```

分析with语句底层发生了什么?

- 调用 `__enter__` 方法， 返回的是文件对象;
- 将文件对象起别名为f;
- 运行完之后， 调用之前暂存 `__exit__` 方法， 实现关闭文件;

In [ ]:

```
```

In [ ]:

```
```

Type *Markdown* and LaTeX: α^2