用Python做些事

05-会自己做事的一类东西

会自己做事的一类东西

- 基本语法
- 属性和封装
- 方法
- 继承和组合
- 多态
- 元编程
- wxpython

基本语法

语法

```
class class_name(base_class):
    class_var
    def methods(self, args):
        statements
```

经典类，新式类

2和3的区别，3都是新式类

经典类和新式类的区别：

1) __slots__,
2) 继承顺序，super
3) __new__,
4) __getattribute__,

```
class A:
    pass

class B（object）：
    pass

a = A()
b = B()
```

会自己做事的一类东西

- ☑ ~~基本语法~~
- ☐ 属性和封装
- ☐ 方法
- ☐ 继承和组合
- ☐ 多态
- ☐ 元编程
- ☐ wxpython

属性和封装

实例和类属性

实例属性
类属性
描述符

__init__

```
class Car(object):
    country = u'中国'

    def __init__(self, length, width, height,
owner=None):
        self.owner = owner
        self.length = length
        self.width = width
        self.height = height
```

属性和封装

私有属性

__xxx

_xxx

__xxx__

封装

```
class Car(object):
    country = u'中国'
    def __init__(self, length, width, height,
owner=None):
        self.__owner = owner
        assert length>0,"length must larger than 0"
        self._length = length
        self._width = width
        self._height = height


    def getLength(self):
        return self._length

    def setLength(self,value):
        assert value>0,"length must larger than 0"
        self._length = value
```

属性和封装

装饰器描述符

```
@propery
@xxx.setter
@xxx.deleter
```

```
class Car(object):
    country = u'中国'
    def __init__(self, length, width, height,
owner=None):
        self._owner = owner
        self._length = length
        self._width = width
        self._height = height

    @property
    def owner(self):
        return self._owner
    @owner.setter
    def owner(self, value):
        self._owner = value
```

属性和封装

__getattr__

__getattr__

__setattr__

__delattr__

```
def __getattr__(self,name):
    print "__getattr__",name
    return self.__dict__.get(name,None)

def __setattr__(self,name,value):
    print "__setattr__",name
    if name!='owner':
        assert value>0, name+" must larger than 0"
    self.__dict__[name]=value

def __delattr__(self,name):
    print "__delattr__",name
    if name=='owner':
        self.__dict__[name]=None
```

属性和封装

描述符

__get__

非数据描述符

__get__
__set__

数据描述符

__get__
__set__

__delete__

```
def __get__(self, instance, owner):
    # instance = x
    # owner = type(x)
    print "__get__",instance
    return self.val

def __set__(self, instance, value):
    # instance = x
    print "__set__",instance
    assert value > 0, "Negative value not
allowed: ”  + str(value)
    self.val = value
```

属性和封装

总结

类属性

实例属性，__getattr__，__setattr__

数据描述符，__get__，__set__，@property

非数据描述符， __get__

__getattribute__

会自己做事的一类东西

- [x] ~~基本语法~~
- [x] ~~属性和封装~~
- [ ] 方法
- [ ] 继承和组合
- [ ] 多态
- [ ] 元编程
- [ ] wxpython

方法

分类

类方法， @classmethod

实例方法

静态方法，@staticmethod

特殊方法（魔法方法），__init__

实质区别：

——绑定类

——绑定实例对象

——无绑定

形式上的区别：
1. 调用是通过类和实例进行的, 不能直接调用.
2. 有自己的特殊参数, self, cls
3. 有自己的声明语法, @classmethod, @staticmethod， __xxx__()

方法

特殊方法

l : less
t : than
e : equal
g : great
n : negative

属性访问：__getattr__, __setattr__, __getattribute__

实例生成 / 类生成：__init__, __new__

数字计算：__add__, __sub__, __mul__, __div__ , __pow__, __round__,

调用方法：__str__, __repr__ , __len__ , __bool__

比较大小：__cmp__, __lt__, __le__, __eq__, __ne__, __gt__, __ge__

集合访问：__setslice__, __getslice__, __getitem__, __setitem__, __contains__

迭代器：__iter__, __next__

会自己做事的一类东西

- ☑ ~~基本语法~~
- ☑ ~~属性和封装~~
- ☑ ~~方法~~
- ☐ 继承和组合
- ☐ 多态
- ☐ 元编程
- ☐ wxpython

继承

简单继承

通过已有的类来生成新类

```
class Employee(object):
    def __init__(self, name, job=None, pay=0):
        self._name = name
        self._job = job
        self._pay = pay
    def giveRaise(self, percent):
        self._pay = int(self._pay * (1 + percent))
    def __str__(self):
        return '[Employee: %s, %s, %s]' % (self._name, self._job,
self._pay)

class Manager(Employee):
    def __init__(self, name, pay):
        Employee.__init__(self, name, 'mgr', pay)
    def giveRaise(self, percent, bonus=.10):
        Employee.giveRaise(self, percent + bonus)
```

继承

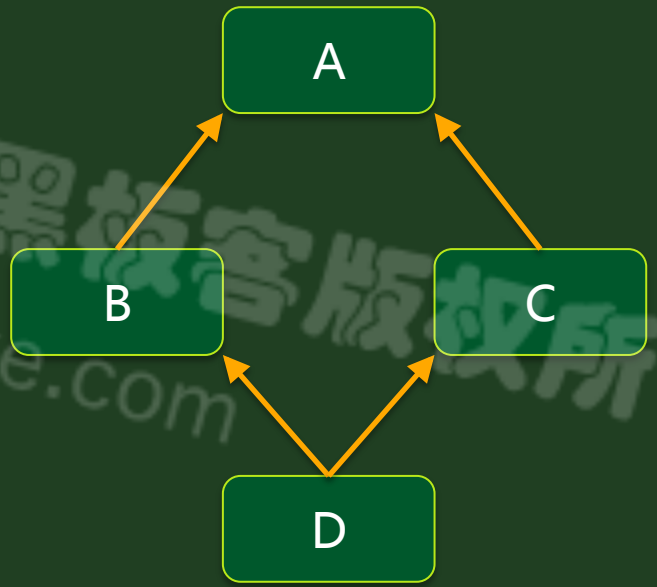
多重继承

MRO
Method
Resolution
Order

Classic：
深度优先

New：
广度优先

```
class A:
    a=1
    b=1

class B(A):
    b=2

class C(A):
    a=3
    b=3
    c=3

class D(B,C):
    pass
```



继承

Super用途

避免父类方法重复调用

Super是一个类，不是函数

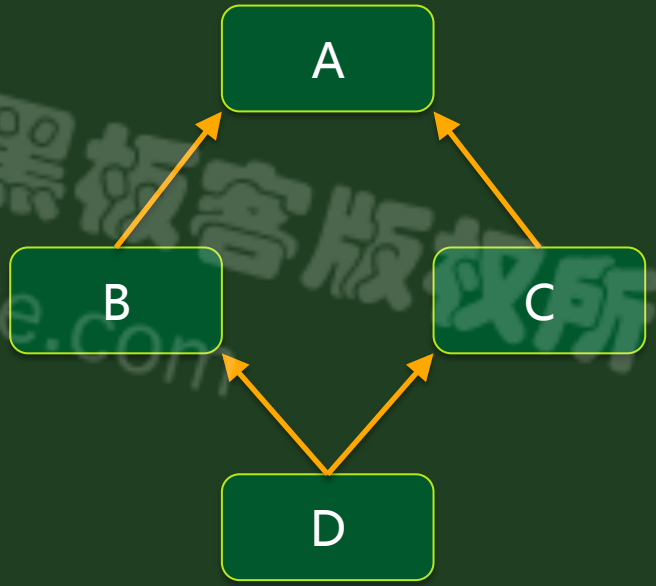
super(D,self).test()

新式类

```
class A:
    def test(self):
        print "A's test"

class B(A):
    def test(self):
        print "B's test"
        A.test(self)

class C(A):
    def test(self):
        print "C's test"
        A.test(self)

class D(B,C):
    def test(self):
        print "D's test"
        B.test(self)
        C.test(self)
```

https://www.python.org/download/releases/2.3/mro/

http://blog.csdn.net/johnsonguo/article/details/585193



组合

has a和is a的区别

通过已有的类来生成新类

```
class Department(object):
    def __init__(self, *args):
        self.members = list(args)

    def addMember(self, person):
        self.members.append(person)

    def showAll(self):
        for person in self.members:
            print person

    def giveRaise(self,percent):
        for person in self.members:
            person.giveRaise(percent)
```

会自己做事的一类东西

- ☑ ~~基本语法~~
- ☑ ~~属性和封装~~
- ☑ ~~方法~~
- ☑ ~~继承和组合~~
- ☐ 多态
- ☐ 元编程
- ☐ wxpython

多态

和重载概念相区分

里氏代换原则：

父类出现的地方，子类一定可以出现，反之则不一定；

```
class Department(object):

    def __init__(self, *args):
        self.members = list(args)
```

Employee，Manager的giveRaise是多态

重载是相同类下相同方法的不同参数类型。对应python是args，kwargs。

多态是不同类的相同方法，相同参数，不同功能。调用时便于将一组对象放在集合里，无需判断对象的具体类型，统一调用。

运算符重载

分类

+, -, *, /
__add__, __sub__, __mul__, __div__

Bool
__bool__

Compare
__lt__, __gt__

Sort
__cmp__

__contains__

运算符重载

遗留问题

1. 如何定义一个三角形
2. 如何为三角形计算面积
3. 如何判断一个点在三角形内

会自己做事的一类东西

- [x] ~~基本语法~~
- [x] ~~属性和封装~~
- [x] ~~方法~~
- [x] ~~继承和组合~~
- [x] ~~多态~~
- [ ] 元编程
- [ ] wxpython

元编程

元编程

面向过程　　语法→语句　　元（Meta）

面向函数　　类→对象　　元类，元编程

面向对象　　元类→类

元编程——type1

Type的用途

Type(name, bases, attrs)

Name: 类名字符串
Bases：父类元组
Attrs：属性字典

```
A = type('A', (object,), {'b': 1})
a = A()
print A, a.b
```

```
def f(name, bases, attrs):
    attrs['c'] = 2
    return type(name, bases, attrs)

A = f('A', (object,), {'b': 1})
```

元编程——type2

Type的用途

```
class SimpleMeta1(type):
    def __init__(cls, name, bases, nmspc):
        super(SimpleMeta1, cls).__init__(name, bases, nmspc)
        cls.uses_metaclass = lambda self : "Yes!"

A=SimpleMeta(  'A'  ,(),{})


 Class A(object):
    __metaclass__=SimpleMeta
```

元编程——元类

元类例子

元类和父类的区别

1. 调用元类的初始化函数，或用metaclass关键字来生成新类
2. 通过继承父类来生成新类

不可继承的类——最终类

```
class final(type):
    def __init__(cls, name, bases, namespace):
        super(final, cls).__init__(name, bases, namespace)
        for klass in bases:
            if isinstance(klass, final):
                raise TypeError(str(klass.__name__) + " is final")
```

元编程——new

New的用途

__new__
__init__

执行顺序

正整数

```
class PositiveInt(int):
    def __new__(cls, value):
        return super(PositiveInt2,cls).__new__(cls, abs(value))
```

单例模式

```
class Singleton(object):
    def __new__(cls):
        if not hasattr(cls, 'instance'):
            cls.instance = super(Singleton, cls).__new__(cls)
        return cls.instance
```

元编程——元类

元类的用法

抽象函数——虚函数，在子类里实现

```
class MyAbstractClass1(object):
    def method1(self):
        raise NotImplementedError("Please Implement this method")
```

接口，由一组抽象函数组成的类

```
from abc import ABCMeta, abstractmethod
class MyAbstractClass2(object):
    __metaclass__ = ABCMeta
    @abstractmethod
    def method1(self):
        pass
```

元编程——ORM

ORM

ORM: 对象关系映射。将数据库的操作用面向对象的程序方法实现。

Object
Relational
Mapping

- 易改：便于更换数据库，sql语句是由底层根据数据库类型生成的，上层数据模型无需变化。
- 易用：便于对数据模型进行操作，创建，更新，查询，删除。用户编写简单，无须写sql语句即可操作数据。
- 易看：使数据模型的程序文档化。便于维护。

元编程——ORM例子

ORM

Object
Relational
Mapping

```
class User(Model):
    id = IntegerField('uid')
    name = StringField('username')
    email = StringField('email')
    password = StringField('password')

u = User(id=12345, name='Michael', email='test@orm.org',
        password='my-pwd')
u.save()
```

会自己做事的一类东西

- [x] ~~基本语法~~
- [x] ~~属性和封装~~
- [x] ~~方法~~
- [x] ~~继承和组合~~
- [x] ~~多态~~
- [x] ~~元编程~~
- [ ] wxpython

wxpython——基础

wx

```
import wx

# 每个wxPython的程序必须有一个wx.App对象.
app = wx.App() #default is False

frame = wx.Frame(None, -1, title=’ Hello World',
pos=(300,400), size=(200,150))
#frame.Centre()
frame.Show()

# 进入循环，等待响应
app.MainLoop()
```

wxpython——画图

PaintDC

```
class Example(wx.Frame):
    def __init__(self, title,shapes):
        super(Example, self).__init__(None, title=title, size=(600, 400))
        self.shapes = shapes
        self.Bind(wx.EVT_PAINT, self.OnPaint)
        self.Centre()
        self.Show()

    def OnPaint(self, e):
        dc = wx.PaintDC(self)
        for shape in self.shapes:
            dc.SetPen(wx.Pen(shape.color))
            dc.DrawLines(shape.drawPoints())
```

wxpython——计算器

Button等控件

- Panel
- BoxSizer / GridSizer
- Textctrl
- Button / Bind
- Event

wxpython——2048

2048

www.7qb.cn/python

会自己做事的一类东西

- ☑ ~~基本语法~~
- ☑ ~~属性和封装~~
- ☑ ~~方法~~
- ☑ ~~继承和组合~~
- ☑ ~~多态~~
- ☑ ~~元编程~~
- ☑ ~~wxpython~~

THANKS

?

作业

- 5-1 如何修改元类的例子meta_04，使其输出更具有级联性
- 5-2 修改wx_01_shape, 画三角形，椭圆，五角星
- 5-3 扩展area计算各个形状的面积
- 5-4 升级wxpython-2048，提供反悔功能

思考题

- 5-5 扩展__contains__，判断点是否在其他Shape中，如三角形，椭圆，五角星等。
- 5-6 升级wxpython-2048，提供帮助功能，让计算机输出最优策略。