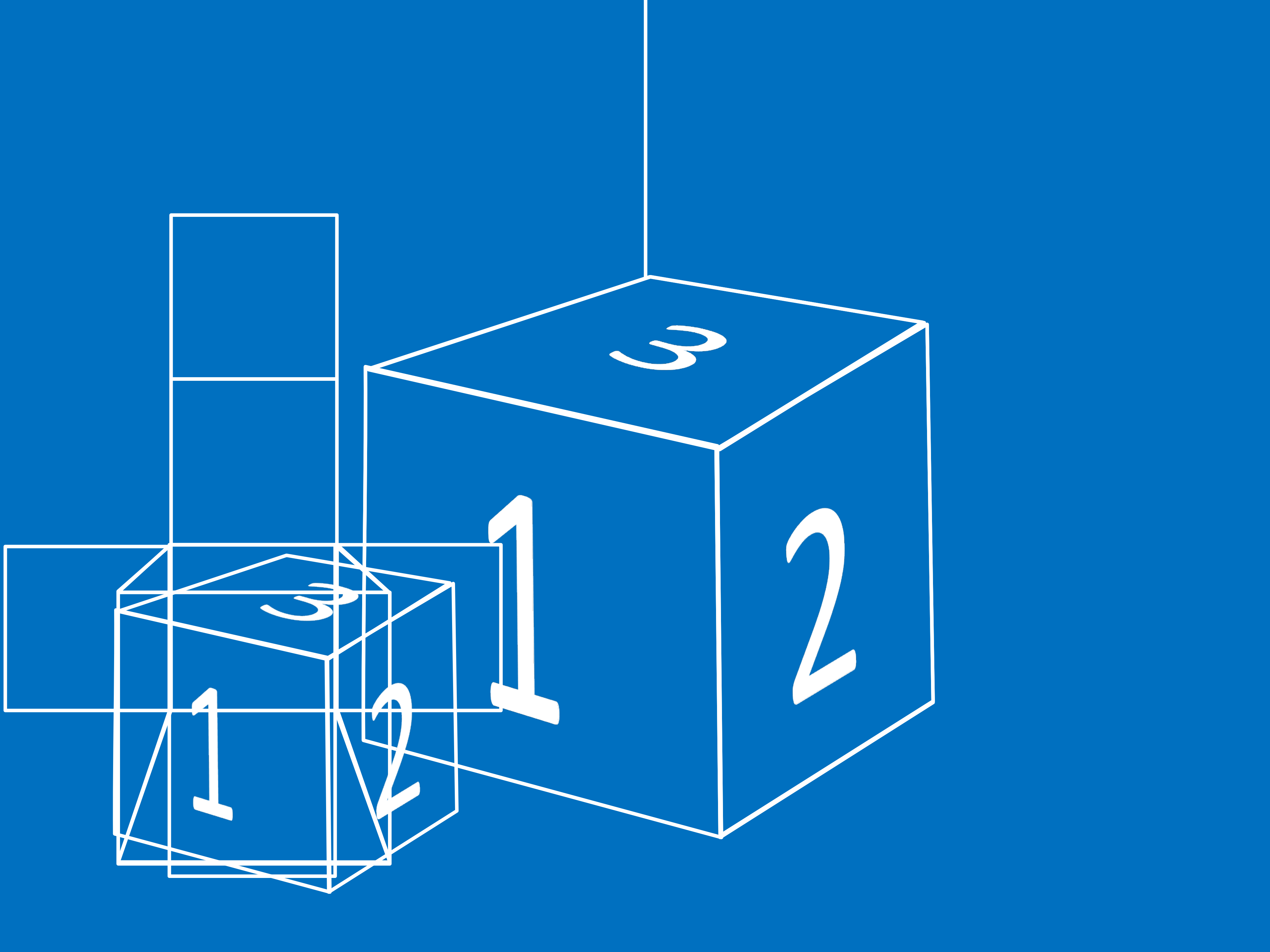
Python基础2

孙鹰


3
1
2
3
1
2

输入输出

数据类型

控制结构

函数定义

异常处理

面向对象

复习

复习内容

Index

- 输入
- 输出
- 文件
- 容器

输入

- In [3]: int_a = input(‘shu ru shu zi:')
- 'shu ru shu zi:233

- In [4]: str_a = input('shu ru zi fu chuan:')
- shu ru zi fu chuan:'spade'

输出

- In [13]: dict1={'food':'banana','cost':24}
- In [14]: print("The %(food)s costs %(cost)7.1f cents" % dict1)

- In [15]: print("%d + %d = %d when you miscalculate it" % (1,1,3))

文件

- In [30]: f=file("D:\\test.txt", "r")
- In [30]: f=file("D:\\test.txt", “w")
- In [30]: f=file("D:\\test.txt", "r+")
- In [30]: f=file("D:\\test.txt", “w+")

赋值过程，基础类型

- a = 6/4
- a = 6//4

- a = 6/4.0
- a = floor(6/4.0)
- a = ceil(6/4.0)
- a = floor(4.0)
- a = ceil(4.0)

四大容器

- collection1=2,3,4,[1,2]，这是什么容器？
- collection2=list(collection1)，collection1=tuple(collection2)，python哲学？
- [1,2] in collection2
- 现在collection2=[2,3,4,[1,2]]
- a = collection2.pop()
- collection2.append(‘phase’)

四大容器

- 复习切片，collection[起:末:步长]，末尾呢？
- collection3=[2,3,4,[1,2]]
- 首先正数步数：
- collection3[1:3]，collection3[:4]，collection3[3:0]
- 接下来是负数步数：
- collection3[::-1]，collection3[-1:-5:-1]还是collection3[-1:-4:-1]

四大容器

- a={}
- set1={ [1,2,2,3,4,4,[3,4]] }
- **d1 = {'a' : 'some value', 'b' : [1, 2, 3, 4]}**
- **d1[25] = 'an integer‘**
- **d1[‘35’] = 'an integer‘**
- **35 in d1?**
- **‘**an integer’ in d1?

四大容器

- d1 = {25: 'integer', ‘35': 'integer', 'a': 'some value', 'b': [1, 2, 3, 4]}
- **d1.keys()**
- **d1.values()**
- **d1.get(‘35’)**

总结

总结

	元组	列表	集合	字典
是否可变				
序列散列				
使用符号				

字符串

字符串

- 字符串string是0个或者多个字符的序列容器
- 使用引号包围，可以是单引号，也可以是双引号，它是不可变的，多行需要三重引号。
- **In [132]: c = """**
- **...: This is a longer string that**
- **...: spans multiple lines**
- **...: """**

- **In [133]: c**
- **Out[133]: '\nThis is a longer string that\nspans multiple lines\n'**

字符串

- 字符串也有属于自己的方法
- aStr.center(w)：返回一个字符串，w长度，原字符串居中
- aStr.count(item)：返回原字符串中出现item的次数
- aStr.ljust(w)：返回一个字符串，w长度，原字符串居左
- aStr.rjust(w)：返回一个字符串，w长度，原字符串居右
- aStr.lower()：返回一个字符串，全部小写
- aStr.upper()：返回一个字符串，全部大写
- aStr.find(item)：查询item，返回第一个匹配的索引位置
- aStr.split(schar)：以schar为分隔符，将原字符串分割，返回一个列表
- 思考：aStr.replace(old, new[, count])

字符串

- In [134]: a = 'this is a string'
-     ...: b = a.replace('string', 'longer string')
-     ...: b
- Out[134]: 'this is a longer string‘

- In [139]: a.split('i')
- Out[139]: ['th', 's ', 's a str', 'ng']

- In [140]: a.split('is')
- Out[140]: ['th', ' ', ' a string']

- 思考：a[3]=‘s’

内置序列函数

内置序列函数

- len用于获得当前容器中所有元素的个数。
- **In [1]: s1={1,33,'sell','buy',(2,3,5,'right')}**
- **In [2]: len(s1)**
- **Out[2]: 5**

- **l1=[2,4,5,'left','right',(6,7,8,'seal')]**

- **len(l1)**
- **Out[6]: 6**
- **len(l1[5])**
- **Out[7]: 4**

内置序列函数

- Enumerate用于跟踪当前序列的索引，产生的结果可以转化为字典。
- **In [174]: some_list = ['foo', 'bar', 'baz']**
- **...: mapping = dict(enumerate(some_list))**
- **In [175]: mapping**
- **Out[175]: {0: 'foo', 1: 'bar', 2: 'baz'}**

内置序列函数

- Sorted可以将任何序列返回为一个有序列表
- In[177]: sorted([7, 1, 2, 6, 0, 3, 2])
- Out[178]: [0, 1, 2, 2, 3, 6, 7]

- 下面这是干什么：
- **In [177]: sorted(set('this is just some string'))**
- **Out[177]: [' ', 'e', 'g', 'h', 'i', 'j', 'm', 'n', 'o', 'r', 's', 't', 'u']**

内置序列函数

- Zip用于将多个序列进行配对，生成元组的列表。也可以进行拆分基友配对。
- **In [179]: seq1 = ['foo', 'bar', 'baz']**
- **...: seq2 = ['one', 'two', 'three']**
- **...: zip(seq1, seq2)**
- **...:**
- **Out[179]: [('foo', 'one'), ('bar', 'two'), ('baz', 'three')]**

- **In [181]: pitchers = [('Nolan', 'Ryan'), ('Roger', 'Clemens'),**
- **...: ('Schilling', 'Curt')]**
- **...: first_names, last_names = zip(*pitchers)**
- **...: first_names**
- **Out[181]: ('Nolan', 'Roger', 'Schilling')**

内置序列函数

- Reversed用于逆序序列中的元素：
- **In [182]: a=list(reversed(range(10)))**
- **Out[182]: a=[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]**

- 输入输出
- 数据类型
- 控制结构
- 函数定义
- 异常处理
- 面向对象

控制结构

Index

- 条件结构
- 循环结构
- 推导式

条件结构

- 回忆：C语言的if和else if
- In [143]: score = 33
- In [145]: if score >= 90:
- ...:   print('A')
- ...: elif score >=80:
- ...:   print('B')
- ...: elif score >= 70:
- ...:   print('C')
- ...: elif score >= 60:
- ...:   print('D')
- ...: else:
- ...:   print('F')
-
- F
- 注意缩进即可

条件结构

- Python中没有switch-case结构
- Python中的三目操作符
- X if C else Y等价于C语言中的 C ? X : Y，更符合外国人的语言习惯。
- x = 3 if 4>=3 else 4
- x
- Out[148]: 3

循环结构

- 回忆：C语言for(int i = 0; i < n; i++)
- for语句用于遍历一个可迭代对象的成员，如列表、字符串等
- for语句结构如下，要强制自己使用！
- for item in collection:
- 具体到C语言的形式为：for i in range(n)
- While，continue，break用法和C语言一样

循环结构

- range()用于产生一组间隔平均的整数
- xrange()有同样的功能，只是不预先产生所有的值，用于数值巨大的时候。
- For远远不止于C中的for
- **In [146]: wordlist = ['cat','dog','rabbit']**
- **...: letterlist = []**
- **...: for aword in wordlist:**
- **...: for aletter in aword:**
- **...: letterlist.append(aletter)**
- **...: letterlist**
- **Out[146]: ['c', 'a', 't', 'd', 'o', 'g', 'r', 'a', 'b', 'b', 'i', 't']**

推导式

- 也叫解析式，为了方便地构建列表、字典等，将条件与表达式浓缩到一句话中。
- 列表推导式：[表达式 + for语句 +(if语句)]
- 将满足条件的值用于计算表达式后得到的值加入到列表中
- **In [149]: strings = ['a', 'as', 'bat', 'car', 'dove', 'python']**
- **...: [x.upper() for x in strings if len(x) > 2]**
- **Out[149]: ['BAT', 'CAR', 'DOVE', 'PYTHON']**

- **In [150]: unique_lengths = {len(x) for x in strings}**
- **...: unique_lengths**
- **Out[150]: {1, 2, 3, 4, 6}**

输入输出

数据类型

控制结构

函数定义

异常处理

面向对象

函数定义

Index

- 函数定义
- 参数类型
- 扩展阅读

函数定义

- 回忆：C语言函数，为什么要设计函数？
- Python的函数定义需要“def”关键字，一个函数名，一组参数，一个函数体，并可能显式返回值
- def function(para1, para2, ...):

```
    ...
    return ...
```

- 一旦return，则跳出该函数体，后面的语句将不再执行

函数定义

- In [151]: def my_function(x, y, z=1.5):
- ...: if z > 1:
- ...: return z * (x + y)
- ...: else:
- ...: return z / (x + y)
- ...:

- In [152]: my_function(5, 6, z=0.7)
- Out[152]: 0.0636363636363636363

- In [153]: my_function(3.14, 7, 3.5)
- Out[153]: 35.49

参数类型

- 大家是否见过func(*args, **kwargs)这样的函数定义，这个*和**让人有点费解。
- 为了方便大家今后查询手册，查询文档，需要介绍python中函数参数的类型。
- 首先，可选参数
- def funcC(a, b=0):

```
def funcC(a, b=0):
    print a
    print b
```

参数类型

- 其次，可变数量参数：
- In [158]: def funcD(a, b, *c):
- ...:   print (a)
- ...:   print (b)
- ...:   print ("length of c is: %d " % len(c))
- ...:   print (c)

- In [159]: funcD(1, 2, 3, 4, 5, 6)
- 1
- 2
- length of c is: 4
- (3, 4, 5, 6)

参数类型

- 再次，位置参数和关键字参数：
- **In [161]: def funcE(a, b, c):**
- **...: print (a)**
- **...: print (b)**
- **...: print (c)**

- **In [162]: funcE(100, 99, 98)**
- **100**
- **99**
- **98**

- **In [163]: funcE(100, c=98, b=99)**
- **100**
- **99**
- **98**

参数类型

- 最后，**的含义
- In [165]: def funcF(a, **b):
-     ...: print (a)
-     ...: for x in b:
-     ...: print (x + ": " + str(b[x]))

- In [166]: funcF(100, c='hello', b=200)
- 100
- c: hello
- b: 200

扩展阅读

- 你可能永远不会用到下面的东西，但是不妨碍你了解他们。
- Lambda函数，闭包，currying生成器，itertools

- 输入输出
- 数据类型
- 控制结构
- 函数定义
- 异常处理
- 面向对象

异常处理

Index

- 异常类型
- 异常处理
- 自建异常

异常类型

- 1、NameError：尝试访问一个未申明的变量
  >>> v

  2、ZeroDivisionError：除数为0
  >>> v = 1/0

  3、SyntaxError：语法错误
  >>> int int

  4、IndexError：索引超出范围
  >>> List = [2]
  >>> List[3]

  5、KeyError：字典关键字不存在
  >>> Dic = {'1':'yes', '2':'no'}
  >>> Dic['3']

异常类型

- 6、IOError：输入输出错误
  >>> f = open('abc')

- 7、AttributeError：访问未知对象属性
  >>> dict1.reverse()

- 8、ValueError：数值错误
  >>> int(‘d’)
  ValueError: invalid literal for int() with base 10: ‘d’

- 9、TypeError：类型错误
  >>> iStr = ’22’   >>> iVal = 22
  >>> obj = iStr + iVal;

- 10、AssertionError：断言错误
  >>> assert 1 != 1

异常处理

- 使用try-except语句，我们可以对异常进行处理
- 语句结构：
- try:

```
try:
    # 可能产生异常的语句
except:
    # 当异常发生时执行的语句
else:
    #try成功时候执行
finally:
    #无论如何都执行
```

异常处理

- f = open(path, 'w')

- try:
- write_to_file(f)
- except:
- print 'Failed'
- else:
- print 'Succeeded'
- finally:
- f.close()

自建异常

- 可以使用raise语句自己抛出异常
- class ResultError(IOError):   pass

- if score>90:
-     raise ResultError('Can\'t be that good!')
- else:
-     print(score)
- ---
- ResultError   Traceback (most recent call last)
- <ipython-input-173-951814d5b258> in <module>()
-       1 if score>90:
- ----> 2   raise ResultError('Can\'t be that good!')
-       3 else:
-       4   print(score)

- ResultError: Can't be that good!

输入输出

数据类型

控制结构

函数定义

异常处理

面向对象

面向对象

Index

- 异常类型
- 异常处理
- 自建异常

面向对象与面向过程

- 算法与数据结构谁先确定？
- 回忆C语言，举个生活中的栗子：
- 如何把衣服放入衣柜？
- 面向过程：为了把衣服装进衣柜，需要3个过程。

  1.把衣柜打开（得到打开的衣柜）

  2.把衣服放进去（打开门后，得到里面存放衣服的衣柜）

  3.把衣柜门关上（打开门、存好衣服、获得关好门的衣柜）
- 每个过程有一个阶段性的目标，依次完成这些过程，就能把衣服存进衣柜。

面向对象与面向过程

- 面向对象：
- 为了把衣服存入衣柜，需要做三个动作（或者叫行为）。
- 每个动作有一个执行者，它就是对象。
- **1)** 衣柜，你给我把门打开
- **2)** 衣柜，你给我把衣服装进去（或者说，衣服，你给我钻到衣柜里去）
- **3)** 衣柜，你给我把门关上
- 依次做这些动作，就能把衣服存入衣柜

  代码如下：
- 衣柜**.**开门**()**
- 衣柜**.**存入**(**衣服**)**
- 衣柜**.**关门**()**

对象和类

person	Sandy	Jack
属性：	属性：	属性：
年龄；	年龄:24	年龄:22
性别；	性别:F	性别:M
身高;	身高:1.78	身高:1.80
体重；	体重: 50	体重: 60
动作：	动作：	动作：
跑；	跑；	跑；
跳；	跳；	跳；
说话；	说话；	说话；

面向对象

- 基本概念
- 面向对象编程的基础是把自己的数据类型定义为类，类是用户定义的数据类型。
- 定义一个类后，就可以用类名声明类类型的变量，即将类实例化为不同的对象。
- 那么，类和对象的区别？

面向对象

- 基本概念
- 创建类时*用变量表示属性*称为成员变量或成员属性，*用函数表示方法*称为成员函数或成员方法，成员属性和成员方法都称为类的成员。

指出变量/属性，函数/方法

person

属性：

年龄；

性别；

身高；

体重；

动作：

跑；

跳；

说话；

Sandy

属性：

年龄:24

性别:F

身高:1.78

体重: 50

动作：

跑；

跳；

说话；

Jack

属性：

年龄:22

性别:M

身高:1.80

体重: 60

动作：

跑；

跳；

说话；

类的定义

- Python使用class保留字来定义类，类名的首字母一般要大写。如，
- class Car:
- def infor(self):
- print(" This is a car ")
- 类的方法至少有一个参数self，self代表将来要创建的对象本身。在类的方法中访问类的实例属性时需要以self为前缀(下面将学习)。

两种属性

- 属性有两种，一种是实例属性，另一种是类属性。实例属性是在构造函数__init__中定义的，定义时以self作为前缀；类属性是在类中方法之外定义的属性。在主程序中（在类的的外部），实例属性属于实例(对象)，只能通过对象名访问；类属性属于类，可通过类名访问。
- 在类的方法中可以调用类的其它方法，可以访问类属性、对象属性。

两种属性

- 【例8-1】 实例属性和类属性。
- class Car:
-     fuel=‘petroleum’   #定义类属性
-     def __init__(self, c):
-         self.color=c   #定义实例属性

- car1=Car("Red")
- car2=Car("Blue")
- print car1.color, Car.fuel
- Car.fuel=‘gas’   #修改类属性
- Car.owner=‘贵族'   #增加类属性
- car1.color="Yellow"   #修改实例属性

访问控制

- 公有属性和私有属性
- 如果属性名以双下划线__开头则是私有属性，否则是公有属性。私有属性在类外不能直接访问。**Python**提供了访问私有属性的方式，可用于程序的测试和调试。
- 私有属性是为了数据封装和保密而设的属性，只能在类的成员方法（称为类的内部）中使用，通过**self**访问。公有属性是可以公开使用的，既可以在内部使用，也可以在外部(通过对象名访问)使用。

访问控制

- 公有方法、私有方法和静态方法。
- 双下划线__开头则是私有方法， 上一行用 @ staticmethod 标识的方法是静态方法，否则是公有方法。
- 公有方法、私有方法都属于对象，每个对象都有自己的公有方法和私有方法；公有方法通过对象名调用，私有方法不能通过对象名调用，只能在属于对象的方法中通过self调用；静态方法属于类，静态方法只能通过类名调用，静态方法中不能访问属于对象的成员，只能访问属于类的成员。

访问控制

```
class Fruit:
    price=0
    def __init__(self):
        self.__color=‘Red’          #定义和设置私有属性color
        self.__city='Kunming'      #定义和设置私有属性city
    def __outputColor(self):       #定义私有方法outputColor
        print(self.__color)        #访问私有属性color
    def __outputCity(self):        #定义私有方法outputCity
       print(self.__city)          #访问私有属性city
```

访问控制

```
class Fruit:
    price=0
    def __init__(self):
        self.__color=‘Red’                #定义和设置私有属性color
        self.__city='Kunming'             #定义和设置私有属性city
    def __outputColor(self):              #定义私有方法outputColor
        print(self.__color)               #访问私有属性color
    def __outputCity(self):               #定义私有方法outputCity
       print(self.__city)                 #访问私有属性city
    def output(self):                     #定义公有方法output
        self.__outputColor( )             #调用私有方法outputColor
        self.__outputCity( )              #调用私有方法outputCity
    @ staticmethod
    def getPrice():                       #定义静态方法getPrice
         return Fruit.price
    @ staticmethod
    def setPrice(p):                      #定义静态方法setPrice
        Fruit.price=p
```

访问控制

主程序：

```
apple=Fruit()
apple.output()
print(Fruit.getPrice( ))
Fruit.setPrice(9)
print(Fruit.getPrice( ))
apple.__outputColor( )
```

构造析构

- Python中类的构造函数是__init__，用来为属性设置初值，在建立对象时自动执行。

- Python中类的析构函数是__del__，用来释放对象占用的资源，在Python收回对象空间之前自动执行。

构造析构

- Python中类的构造函数是__init__，用来为属性设置初值，在建立对象时自动执行。

- Python中类的析构函数是__del__，用来释放对象占用的资源，在Python收回对象空间之前自动执行。

构造析构

```
class Car :
    def __init__(self, n):
        self.num=n
        print ('编号为', self.num, '的车出生了...')
    def __del__(self):
        print('编号为', self.num, '的车死了...')
car1=Car(1)
car2=Car(2)
del car1
del car2
```

运算符重载

方法	重载	说明	调用举例
__add__	+	对象加法	Z=X+Y，X+=Y
__sub__	-	对象减法	Z=X-Y
__mul__	*	对象乘以实数	Z=X*f
__div__	/	对象除以实数	Z=X/f
__le__	比较	对象比较	X==Y，X!=Y
__lt__	比较	对象比较	X >Y，X<Y, X>=Y, X<=Y
__str__	输出	输出对象时调用	print X
get_len	长度	对象长度	len(X)

运算符重载

例子：定义一个类Vector2代表二维向量，使用运算符重载完成向量的常用运算。以加为例

```
import math
class Vector2(object):
    def __init__(self, x=0.0, y=0.0):
        (self.x,self.y)=(x,y)
    def __str__(self):    #按元组输出
            return "(%s, %s )" %(self.x, self.y)
    def __add__(self,other):  #向量的加法
        third=Vector2()
        third.x=self.x+other.x
        third.y=self.y+other.y
        return third
```

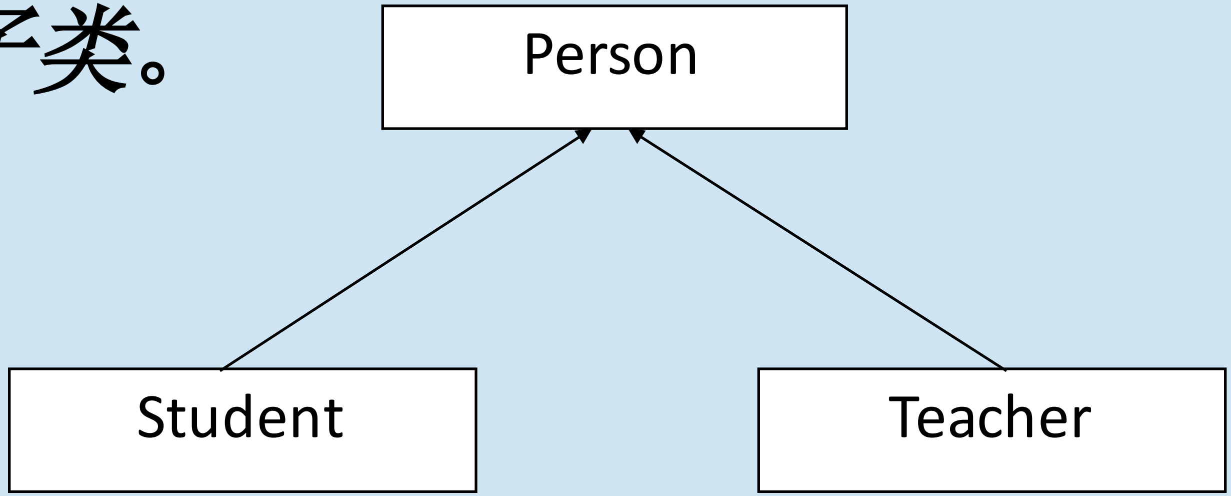
运算符重载

例子：定义一个类Vector2代表二维向量，使用运算符重载完成向量的常用运算。

```
class Vector2(object):

    …
v1=Vector2(1,2)
v2=Vector2(3,4)
v3=v1+v2
print v3
```

继承

- 继承是为代码重用而设计的，当我们设计一个新类时，为了代码重用可以继承一个已设计好的类。在继承关系中，原来设计好的类称为父类，新设计的类称为子类。



类的继承关系,父类及两个子类

继承

- 继承关系可以被传递，如果c1是c2的子类，c2是c3的子类，那么c1也是c3的子类。
- 如果a继承b， 那么a具有b的一切属性和方法。
- Python支持多继承
  - class A:
  - def __init__(*self*):
  - *pass*
  - class B:
  - def __init__(*self*):
  - *pass*
  - class C(A,B):
  - def __init__(*self*):
  - *pass*

继承

- 多态：
- 向不同的对象发送同一个消息，不同的对象在接收时会产生不同的行为(即方法)。
- **Python** 中的多态并没有完全实现，**Python**中只是利用多元化来实现部分多态的特性
- **super**关键字：调用父类的函数

继承

- class Person(object) :
-     role=’ordinary’
-     def __init__(self, s1, s2):
-         self.IDcard=s1
-         self.name=s2
-         print (’constructor of Person’)
-     def greetings(self, friend):
-         print (’hello, ’, friend,’!’)

继承

```
class Student(Person):
    def __init__(self, num):
        self.number=num
    def info(self):
         print (Student.role, self.IDcard, self.name, self.number)

    def greetings(self, friend):
        print ( ‘\” hello, ’, friend,’!\n I am a student!” ’,sep=’’)

class Teacher(Person):
    def __init__(self, t, s1, s2):
        self.title=t
        super(Teacher,self).__init__(s1, s2)    #调用父类的构造函数
    def info(self):
        print (self.role, self.IDcard, self.name, self.title)

    def greetings(self, friend):
        print (’\” hello, ’, friend, ’!\n Time to hand in 
homework\” ’,sep=’’)
```

继承

- 主程序：
- **Student.role = 'Student'**
- **stud1=Student('2012009')**
- **stud1.IDcard='533001198006250079'**
- **stud1.name='Tan Xiaofeng'**
- **stud1.info()**
- **stud1.greetings('Song Rentou')**

- **print('')**

- **Teacher.role='Teacher'**
- **teach1=Teacher('professor', '533001197012130055', ‘Mao Zexi')**
- **teach1.info()**
- **teach1.greetings('Yao Guake')**

继承

- 输出：
- Student 533001198006250079 Tan Xiaofeng 2012009
- “hello, Song Rentou!
- I am a student!”

- constructor of Person
- Teacher 533001197012130055 Mao Zexi professor
- “hello, Yao Guake!
- Time to hand in homework”

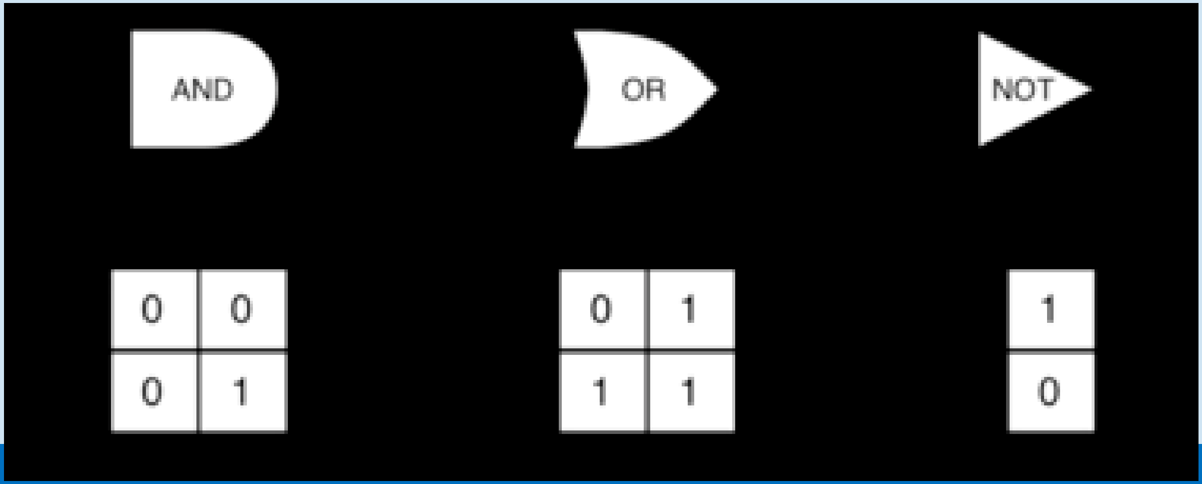
继承

- 输出：
- Student 533001198006250079 Tan Xiaofeng 2012009
- “hello, Song Rentou!
- I am a student!”

- constructor of Person
- Teacher 533001197012130055 Mao Zexi professor
- “hello, Yao Guake!
- Time to hand in homework”

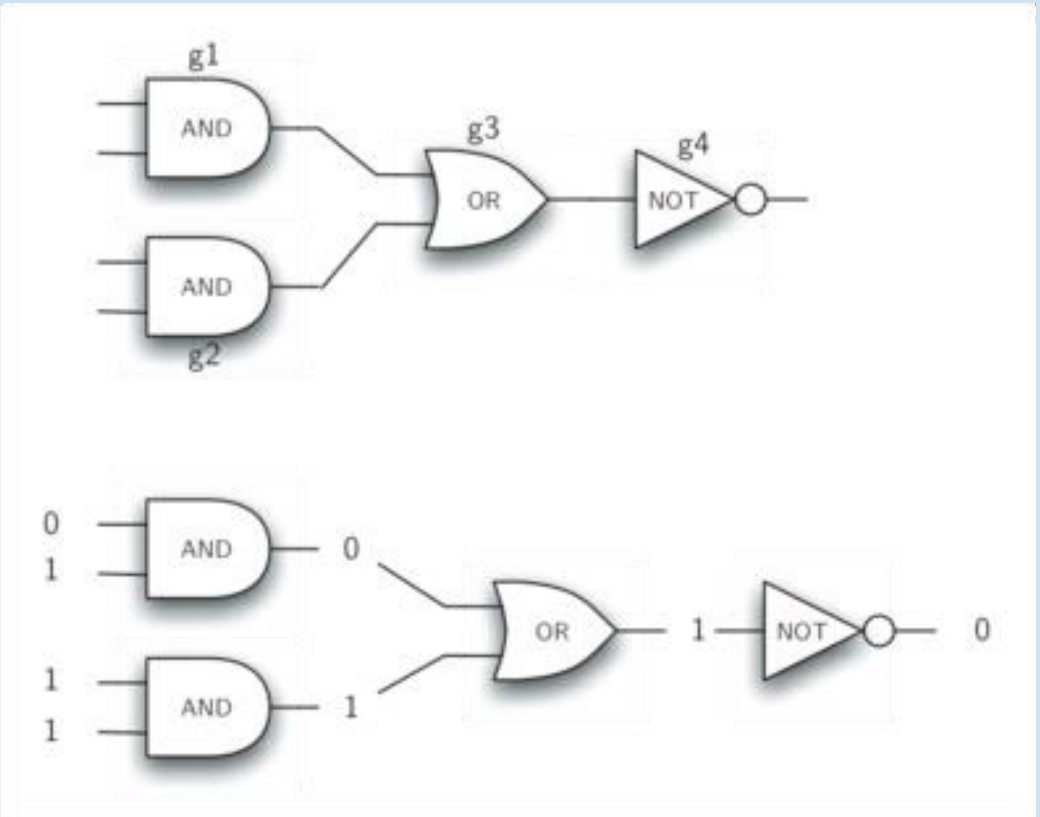
继承

- **第二个高难度的例子：**
- 组合逻辑电路是由逻辑门连接组成，有固定数目的输入和输出，执行某种函数功能。输入输出为布尔值，取自集合{0, 1}，其中0表示FALSE（假），1表示TRUE（真）。
- 与门、或门、非门及其真值表



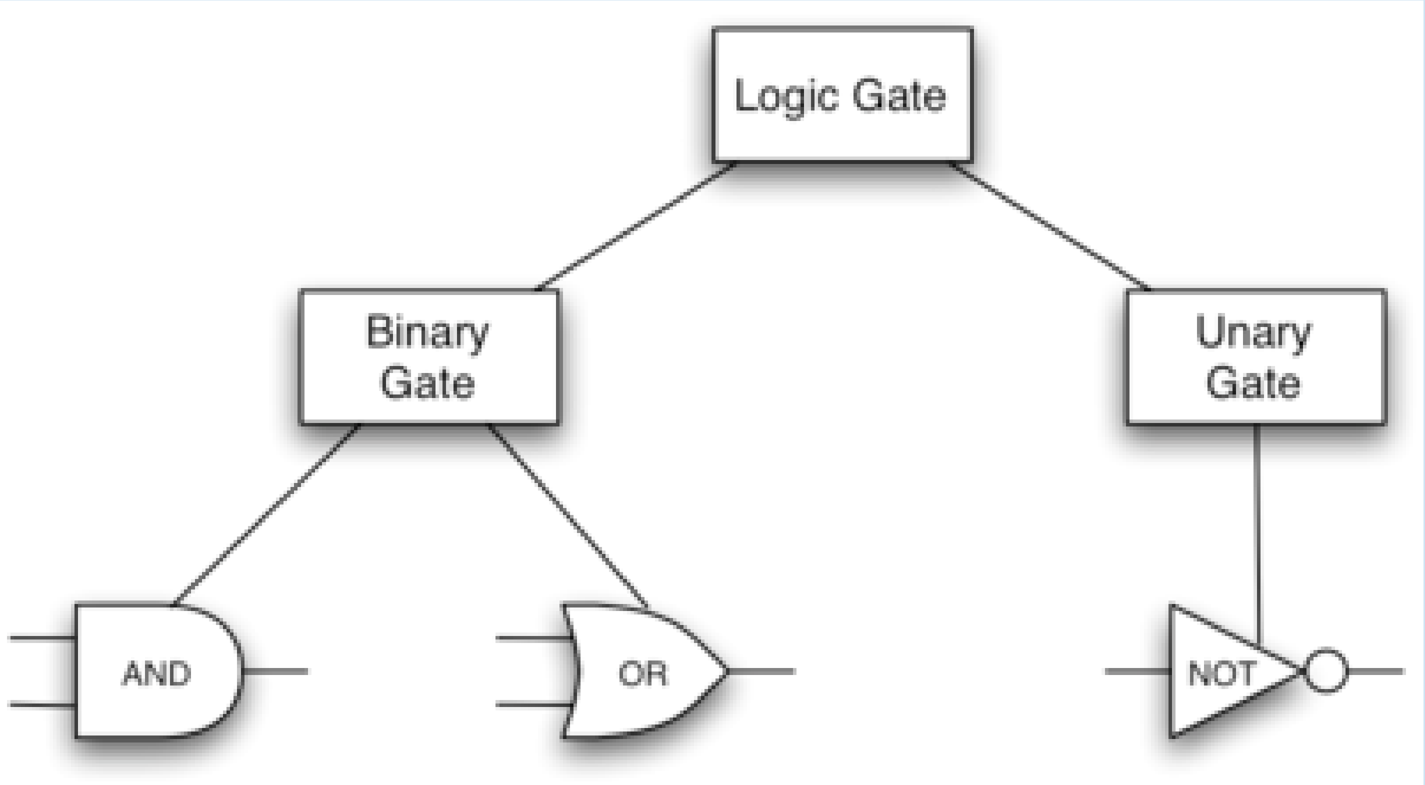
继承

- 一个实际的组合逻辑电路：
- 四个输入，一个输出



继承

- 逻辑门的抽象
- 二元逻辑电路/一元逻辑电路



继承

- 逻辑门——LogicGate类
- 属性：label, output
- 方法：getLabel, getOutput, performGateLogic

```python

class LogicGate(object):
    def __init__(self,n):
        self.label = n
        self.output = None

    def getLabel(self):
        return self.label

    def getOutput(self):
        self.output = self.performGateLogic()
        return self.output

```

继承

- 一元逻辑门——UnaryGate类
- 属性：pin
- 方法：getPin

```python

class UnaryGate(LogicGate):

    def __init__(self, n):
        LogicGate.__init__(self, n)

        self.pin = None

    def getPin(self):
        return int(input("Enter Pin input for gate "
            + self.getLabel() + "-->"))

```

继承

- 二元逻辑门——BinaryGate类
- 属性：pinA, pinB
- 方法：getPinA, getPinB

```python

class BinaryGate(LogicGate):

    def __init__(self, n):
        LogicGate.__init__(self, n)

        self.pinA = None
        self.pinB = None

    def getPinA(self):
        return int(input("Enter Pin A input for gate "
            + self.getLabel()+"-->"))

    def getPinB(self):
        return int(input("Enter Pin B input for gate "
            + self.getLabel()+"-->"))

```

继承

- 非门——NotGate类
- 方法：performGateLogic

```python
class NotGate(UnaryGate):

    def __init__(self, n):
        UnaryGate.__init__(self, n)

    def performGateLogic(self):
        p = self.getPin()
        if p == 1:
            return 0
        else:
            return 1
```

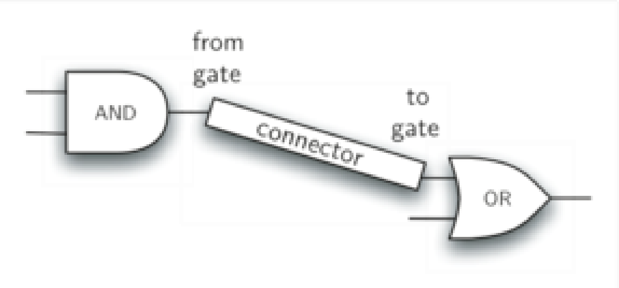
继承

- 与门AndGate和或门OrGate（自行完成）
- 属性：label, output
- 方法：getLabel, getOutput, performGateLogic

```python

class AndGate(BinaryGate):

    def __init__(self, n):
        BinaryGate.__init__(self, n)

    def performGateLogic(self):
        a = self.getPinA()
        b = self.getPinB()
        if a == 1 and b == 1:
            return 1
        else:
            return 0

```

继承

- 将逻辑门组合起来——连接器Connector类




```python
class Connector:

    def __init__(self, fgate, tgate):
        self.fromgate = fgate
        self.togate = tgate
        tgate.setNextPin(self)

    def getFrom(self):
        return self.fromgate

    def getTo(self):
        return self.togate
```

继承

- UnaryGate实现
- 利用之前未使用的状态pin表示输入端连接的Connector

```python

class UnaryGate(LogicGate):
    def setNextPin(self, source):
        if self.pin == None:
            self.pin = source
        else:
            raise RuntimeError('Error: no empty pins')
    def getPin(self):
        if self.pin == None:
            return int(input("Enter Pin input for gate "
                + self.getLabel() + "-->"))
        else:
            return self.pin.getFrom().getOutput()

```

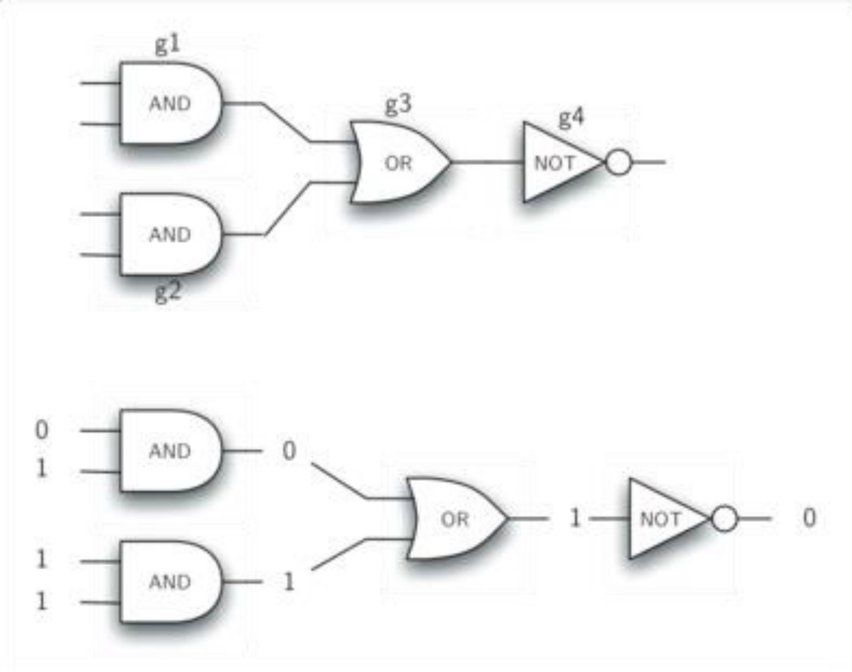
继承

- BinaryGate实现
- 同理，使用pinA和pinB

```python

class BinaryGate(LogicGate):
    def setNextPin(self, source):
        if self.pinA == None:
            self.pinA = source
        elif self.pinB == None:
            self.pinB = source
        else:
            raise RuntimeError('Error: no empty pins')
    def getPinA(self):
        if self.pinA == None:
            return int(input("Enter Pin A input for gate "
                + self.getLabel() + "-->"))
        else:
            return self.pinA.getFrom().getOutput()

```

继承

- 最后大功告成




```
>>> g1 = AndGate("G1")
>>> g2 = AndGate("G2")
>>> g3 = OrGate("G3")
>>> g4 = NotGate("G4")
>>> c1 = Connector(g1, g3)
>>> c2 = Connector(g2, g3)
>>> c3 = Connector(g3, g4)
>>> g4.getOutput()
Enter Pin A input for gate G1-->0
Enter Pin B input for gate G1-->1
Enter Pin A input for gate G2-->1
Enter Pin B input for gate G2-->1
0
```

DS&A

Thank You

designed by Sun