Python语言项目教学05

陈斌

北京大学地球与空间科学学院

2018.11.09

Python语言项目教学培训课程总体安排

日期	上午	下午
Day1	信息技术为基础的创新教育	Python语言概览和教学实践交流
Day2	Python 语言基本数据类型	输入输出、控制流和程序结构
Day3	扩展模块（时间/算术/持久化/数据库）	高级特性（面向对象/异常处理/迭代器/生成器等）
Day4	扩展模块（数值计算/网络/可视化）	Python科学编程实践项目
Day5	Python艺术编程教学	艺术编程实践项目
Day6	Python开源硬件基础	开源硬件实验
Day7	Python开源硬件编程教学	开源硬件编程实践项目
Day8	实习项目分组讨论、开发	编程开发、展示和总结

北京大学 陈斌 gischen@pku.edu.cn

面向对象：什么是对象？

- Python中的所有事物都是以对象形式存在
  - 从简单的数值类型，到复杂的代码模块，都是对象。
- 对象以id作为标识，既包含数据（属性），也包含代码（方法）
  - 赋值语句给予对象以名称，对象可以有多个名称（变量引用），但只有一个id
  - 同一类（class）的对象具有相同的属性和方法，但属性值和id不同
- 对象实现了属性和方法的封装，是一种数据抽象机制

面向对象：类的定义与调用

- 类是对象的模版，封装了对应现实实体的性质和行为
- 定义类：class语句；
  - class <类名>:
  - def __init__(self, <参数表>):
  - def <方法名>(self, <参数表>):
- 调用类：<类名> (<参数>)
  - 调用类会创建一个对象，（注意括号！）
  - obj = <类名> (<参数表>)
  - 返回一个对象实例，
  - 类方法中的self指这个对象实例！

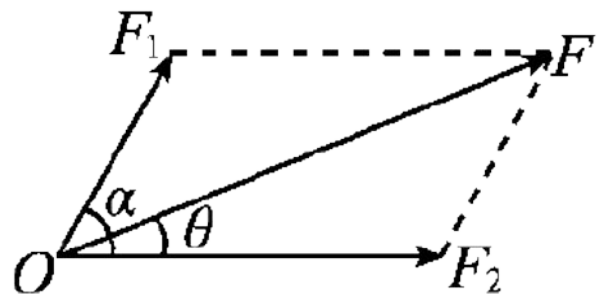
```python
class Force:  # 力
    def __init__(self, x, y):  # x,y方向分量
        self.fx, self.fy = x, y

    def show(self):  # 打印出力的值
        print("Force<%s,%s>" % (self.fx, self.fy))

    def add(self, force2):  # 与另一个力合成
        x = self.fx + force2.fx
        y = self.fy + force2.fy
        return Force(x, y)


# 生成一个力对象
f1 = Force(0, 1)
f1.show()

# 生成另一个力对象
f2 = Force(3, 4)
# 合成为新的力
f3 = f1.add(f2)
f3.show()
```




```
Force<0,1>
Force<3,5>
```

对象属性和方法的引用

- 通过<对象名>.<属性名>的形式引用，可以跟一般的变量一样用在赋值语句和表达式中
- Python语言动态的特征，使得对象可以随时增加或者删除属性或者方法
  - 也必须先赋值再引用

```
44  print(f3.fx, f3.fy)
45  f3.fz = 3.4
46  print(f3.fz)
47  del f3.fz
```

```
0.0 4.5
3.4
```

类定义中的特殊方法

- 在类定义中实现一些特殊方法，可以方便地使用python一些内置操作
  - 所有特殊方法以两个下划线开始结束
  - __str__(self): 自动转换为字符串
  - __add__(self, other): 使用+操作符
  - __mul__(self, other): 使用*操作符
  - __eq__(self, other): 使用==操作符
- 其它特殊方法参见课程网站
  - http://gis4g.pku.edu.cn/python-magic-method/

```
13      __add__ = add
14
15      def __str__(self):
16          return "F<%s,%s>" % (self.fx, self.fy)
17
18      def __mul__(self, n):
19          x, y = self.fx * n, self.fy * n
20          return Force(x, y)
21
22      def __eq__(self, force2):
23          return (self.fx == force2.fx) and \
24                 (self.fy == force2.fy)
```

```
37  # 操作符使用
38  f3 = f1 + f2
39  print("Fadd=%s" % (f3,))
40  f3 = f1 * 4.5
41  print("Fmul=%s" % (f3,))
42  print("%s==%s? -> %s" % (f1, f2, f1 == f2))
```

```
Fadd=F<3,5>
Fmul=F<0.0,4.5>
F<0,1>==F<3,4>? -> False
```

自定义对象的排序

- Python列表类型的sort方法和内置排序函数sorted()
  - 每种数据类型可以定义特殊方法def __lt__(self, y)
  - 返回True视为比y“小”，排在前，而返回False视为比y“大”，排在后
  - 任何自定义类都可以使用x<y这样的比较，只要类中定义了特殊方法__lt__
- 例子：Student
  - 姓名，成绩
- 按照成绩排序
  - 由高到低
- 用内置sort

```
class Student:
    def __init__(self, name, grade):
        self.name, self.grade = name, grade

    # 内置sort函数只引用 < 比较符来判断前后
    def __lt__(self, other):
        # 成绩比other高的，排在他前面
        return self.grade > other.grade

    # Student的易读字符串表示
    def __str__(self):
        return "(%s,%d)" % (self.name, self.grade)

    # Student的正式字符串表示，我们让它跟易读表示相同
    __repr__ = __str__
```

Python可扩展的“大小”比较及排序

- 我们构造一个Python列表
- 在列表中加入Student对象
- 直接调用列表的sort方法
- 可以看到已经根据__lt__定义排序
- 直接检验Student对象的大小
  - <
- 另外可以定义其它比较符
  - __gt__等

```
# 构造一个Python List对象
s = list()

# 添加Student对象到List中
s.append(Student("Jack", 80))
s.append(Student("Jane", 75))
s.append(Student("Smith", 82))
s.append(Student("Cook", 90))
s.append(Student("Tom", 70))
print("Original:", s)

# 对List进行排序，注意这是内置sort方法
s.sort()

# 查看结果，已经按照成绩排好序
print("Sorted:", s)
```

```
============= RESTART: /Users/chenbin/Documents/homework/stu.py =
Original: [(Jack,80), (Jane,75), (Smith,82), (Cook,90), (Tom,70)]
Sorted: [(Cook,90), (Smith,82), (Jack,80), (Jane,75), (Tom,70)]
>>> s[0]<s[1]
True
>>>
```

Python可扩展的“大小”比较及排序

- 我们可以把__lt__方法重新定义，改为比较姓名
- 这样sort方法就能按照姓名来排序

```
class Student:
    def __init__(self, name, grade):
        self.name, self.grade = name, grade

    # 内置sort函数只引用 < 比较符来判断前后
    def __lt__(self, other):
        # 姓名字母顺序在前，就排在他前面
        return self.name < other.name

    # Student的易读字符串表示
    def __str__(self):
        return "(%s,%d)" % (self.name, self.grade)

    # Student的正式字符串表示，我们让它跟易读表示相同
    __repr__ = __str__
```

```
============= RESTART: /Users/chenbin/Documents/homework/stu2.py =
Original: [(Jack,80), (Jane,75), (Smith,82), (Cook,90), (Tom,70)]
Sorted: [(Cook,90), (Jack,80), (Jane,75), (Smith,82), (Tom,70)]
>>> s[0]<s[1]
True
>>>
```

类的继承机制：代码复用

- 如果两个类具有“一般-特殊”的逻辑关系，那么特殊类就可以作为一般类的“子类”来定义，从“父类”继承属性和方法
  - `class <子类名>(<父类名>):`
  - `    def <重定义方法>(self,…):`
- 子类对象可以调用父类方法，除非这个方法在子类中重新定义了（覆盖override）

类继承例子

```
71  gcar=GasCar("BMW")
72  gcar.fill_fuel(50.0)
73  gcar.run(200.0)
```

```
75  ecar=ElecCar("Tesla")
76  ecar.fill_fuel(60.0)
77  ecar.run(200.0)
```

```
BMW: run 200 miles!
Tesla: fuel out!
```

```
45  class Car:
46      def __init__(self, name):
47          self.name = name
48          self.remain_mile = 0
49
50      def fill_fuel(self, miles):  # 加燃料里程
51          self.remain_mile = miles
52
53      def run(self, miles):  # 跑miles英里
54          print (self.name, end=': ')
55          if self.remain_mile >= miles:
56              self.remain_mile -= miles
57              print("run %d miles!" % (miles,))
58          else:
59              print("fuel out!")
60
61
62  class GasCar(Car):
63      def fill_fuel(self, gas):  # 加汽油gas升
64          self.remain_mile = gas * 6.0  # 每升跑6英里
65
66
67  class ElecCar(Car):
68      def fill_fuel(self, power):  # 充电power度
69          self.remain_mile = power * 3.0  # 每度电3英里
```

子类与父类

- 子类可以添加父类中没有的方法和属性
- 如果子类同名方法覆盖了父类的方法，仍然还可以调用父类的方法

```
class GasCar(Car):
    def __init__(self, name, capacity):   # 名称和排量
        super().__init__(name)   # 父类初始化方法，只有名称
        self.capacity = capacity   # 增加了排量属性
```

关于self

- 在类定义中，所有方法的首个参数一般都是self
- self实际上代表对象实例
  - <对象>.<方法>(<参数>)
- 等价于：
  - <类>.<方法>(<对象>, <参数>)
- 这里的对象就是self了
- 如右图Line81和82

```
79  gcar = GasCar("BMW")
80  gcar.fill_fuel(50.0)
81  gcar.run(200.0)
82  GasCar.run(gcar, 200.0)
```

上机练习

- 创建一个类People
  - 包含属性name，city
  - 可以转换为字符串形式（__str__）
  - 包含方法moveto(self, newcity)
  - 可以按照city排序
  - 创建4个人对象，放到列表进行排序
- 创建一个类Teacher
  - 是People的子类，新增属性school
  - moveto方法改为newschool
  - 按照school排序
  - 创建4个教师对象，放到列表进行排序
- 创建一个mylist类，继承自内置数据类型list（列表）
  - 增加一个方法“累乘”product
  - def product(self):
  - 返回所有数据项的乘积。

例外处理Exception

- 代码运行可能会意外各种错误：
  - 语法错误：Syntax Error
  - 除以0错误：ZeroDivisionError
  - 列表下标越界：IndexError
  - 类型错误：TypeError…
- 事先无法预料，如：
  - 由用户输入/交互引起
  - 由外部数据引起
  - 由设备连接等引起

```
>>> lst=[1,2,3]
>>> for i in range(4):
        print(lst[i])


1
2
3
Traceback (most recent call last):
  File "<pyshell#178>", line 2, in <module>
    print(lst[i])
IndexError: list index out of range
>>>
```

```
>>> c=int(input("Please input number:"))
Please input number:ABCD
Traceback (most recent call last):
  File "<pyshell#167>", line 1, in <module>
    c=int(input("Please input number:"))
ValueError: invalid literal for int() with base 10: 'ABCD'
>>>
```

例外处理Exception Handling

- 错误会引起程序中止退出
- 如果希望掌控意外，就需要在可能出错误的地方设置陷阱捕捉错误
  - try: # 为缩进的代码设置陷阱
  - except: # 处理错误的代码
  - else: # 没有出错执行的代码
  - finally: # 无论出错否，都执行的代码

```
1   try:
2       print('try...')
3       r = 10 / 'xyz'
4       print('result:', r)
5   except TypeError as e:
6       print('TypeError:', e)
7   except ZeroDivisionError as e:
8       print('ZeroDivisionError:', e)
9   else:
10      print('no error!')
11  finally:
12      print('finally...')
13  print('END')
```

```
try...
TypeError: unsupported operand type(s) for /: 'int' and 'str'
finally...
END
```

推导式

- 可以用来生成列表、字典和集合的语句
  - [<表达式> for <变量> in <可迭代对象> if <逻辑条件>]
  - {<键值表达式>:<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>}
  - {<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>}

```
>>> [x*x for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
>>> {'K%d'%(x,):x**3 for x in range(10)}
{'K2': 8, 'K8': 512, 'K5': 125, 'K6': 216, 'K3': 27, 'K9': 729, 'K0': 0,
'K7': 343, 'K1': 1, 'K4': 64}
>>>
>>> {x*x for x in range(10)}
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
>>>
>>> {x+y for x in range(10) for y in range(x)}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17}
```

推导式

```
>>> [x+y for x in range(10) for y in range(x)]
[1, 2, 3, 3, 4, 5, 4, 5, 6, 7, 5, 6, 7, 8, 9, 6, 7, 8, 9, 10, 11, 7, 8, 9
, 10, 11, 12, 13, 8, 9, 10, 11, 12, 13, 14, 15, 9, 10, 11, 12, 13, 14, 15
, 16, 17]
>>>
>>> [x*x for x in range(10) if x % 2 == 0]
[0, 4, 16, 36, 64]
>>>
>>> [x.upper() for x in [1, 'abc', 'xyz', True] if isinstance(x, str)]
['ABC', 'XYZ']
```

生成器推导式

- 与推导式一样语法：
  - (<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>)
- 返回一个生成器对象，也是可迭代对象
- 但生成器并不立即产生全部元素，仅在要用到元素的时候才生成，可以极大节省内存

```
>>> agen = (x*x for x in range(10))
>>> agen
<generator object <genexpr> at 0x1078f5620>
>>> for n in agen:
        print (n)


0
1
4
9
16
25
36
```

生成器函数

- 如果生成器较复杂，一行表达式无法容纳，可以定义生成器函数
- 生成器函数的定义与普通函数相同，只是将return换成了yield
  - yield会立即返回一个值
  - 但在下一次迭代生成器函数的时候，会从yield语句后的语句继续执行，直到再次yield返回，或终止
  - return语句则不同，它会终止函数的执行，下次调用会重新执行函数

```
def even_number(max):
    n = 0
    while n < max:
        yield n
        n += 2

for i in even_number(10):
    print (i)
```

```
============= RESTART:
0
2
4
6
8
>>>
```

北京大学 陈斌 gischen@pku.edu.cn

上机练习

- 编写程序，输入两个数，输出它们的商，采用例外处理来处理两种错误，给出用户友好的提示信息
  - 1）除数为0
  - 2）输入了非数值
- 编写一个推导式，生成包含100以内所有勾股数(i,j,k)的列表
- 编写一个生成器函数，能够生成斐波那契数列

```
def fib():
    ….
for fn in fib():
    print (fn)
    if fn>1000:
        break
```