Python 程序设计教案

<table>
<tr><td>本次授课内容</td><td colspan="2">6.1 类的定义与使用
6.2 数据成员与成员方法
6.3 继承
6.4 特殊方法
6.5 综合案例解析</td></tr>
<tr><td>本次课的
教学目的</td><td colspan="2">掌握类的定义语法
掌握对象的创建语法
理解数据成员与成员方法的区别
理解私有成员与公有成员的区别
理解属性的工作原理
了解继承的基本概念
了解特殊方法的概念与工作原理</td></tr>
<tr><td>本次课教学
重点与难点</td><td colspan="2">私有成员与公有成员的区别
属性工作原理
特殊方法的用法</td></tr>
<tr><td>教学方法
教学手段</td><td colspan="2">PPT、边讲边练</td></tr>
<tr><td rowspan="3">课堂教学
时间分配</td><td>教学内容</td><td>时间分配（分）</td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td>课堂教学设计</td><td colspan="2">首先介绍教材上的基本概念和语法，然后重点演示私有成员的定义和特殊方法的应用。</td></tr>
<tr><td>实　验</td><td colspan="2">教材例题 6-1 到 6-4</td></tr>
<tr><td>思考题及作业题</td><td colspan="2">本章所有课后习题</td></tr>
<tr><td>备　注</td><td colspan="2"></td></tr>
<tr><td>教学后记</td><td colspan="2"></td></tr>
<tr><td></td><td colspan="2">第　一　节　课</td></tr>
</table>

课堂重点内容详解	Python 使用关键字 class 来定义类，之后是一个空格，接下来是类的名字，如果派生自其他基类的话则需要把所有基类放到一对圆括号中并使用逗号分隔，然后是一个冒号，最后换行并定义类的内部实现。其中，类名最好与所描述的事物有关，且首字母一般要大写。例如：
	
	`class Car(object):  #定义一个类，派生自 object 类`
	`    def showInfo(self):  #定义成员方法`
	`        print("This is a car")`
	
	定义了类之后，就可以用来实例化对象，并通过“对象名.成员”的方式来访问其中的数据成员或成员方法。例如：
	
	`>>> car = Car()  #实例化对象`
	`>>> car.showInfo()  #调用对象的成员方法`
	`This is a car`
	
	从形式上看，在定义类的成员时，如果成员名以两个下划线开头但是不以两个下划线结束则表示是私有成员。私有成员在类的外部不能直接访问，一般是在类的内部进行访问和操作，或者在类的外部通过调用对象的公有成员方法来访问，而公有成员是可以公开使用的，既可以在类的内部进行访问，也可以在外部程序中使用。
	
	要注意的是，Python 并没有对私有成员提供严格的访问保护机制，通过一种特殊方式“对象名._类名__xxx”也可以在外部程序中访问私有成员，但不建议这样做。
	
	`>>> class Test:`
	`    def __init__(self, value=0): #构造方法，创建对象时自动调用`
	`        self.__value = value  #私有数据成员`
	
	`    def setValue(self, value):  #公有成员方法，需要显式调用`
	`        self.__value = value  #在类内部可以直接访问私有成员`
	
	`    def show(self):  #成员方法，公有成员`
	`        print(self.__value)`
	
	`>>> t = Test()`
	`>>> t.show()  #在类外部可以直接访问非私有成员`
	`0`
	`>>> t._Test__value  #在外部使用特殊形式访问私有数据成员`
	`0`
	
	Python 类的成员方法大致可以分为公有方法、私有方法、静态方法和类方法。公有方法和私有方法一般是指属于对象的实例方法，其中私有方法的名字以两个下划线“__”开始。公有方法通过对象名直接调用，私有方法不能通过对象名直接调用，只能在其他实例方法中通过前缀 self 进行调用或在外部通过特殊的形式来调用。
	
	所有实例方法都必须至少有一个名为 self 的参数，并且必须是方法的第一个形参（如果有多个形参的话），self 参数代表当前对象。在实例方法中访问实例成员时需要以 self 为前缀，但在外部通过对象名调用对象方法时并不需要传递这个参数，因为通过对象调用公有方法

时会把对象隐式绑定到 self 参数。

静态方法和类方法都可以通过类名和对象名调用，但在这两种方法中不能直接访问属于对象的成员，只能访问属于类的成员。类方法一般以 cls 作为类方法的第一个参数表示该类自身，在调用类方法时不需要为该参数传递值，而静态方法则可以不接收任何参数。例如：

```
>>> class Root:
    __total = 0
    def __init__(self, v):      #构造方法，特殊方法
        self.__value = v
        Root.__total += 1

    def show(self):             #普通实例方法，以 self 作为第一个参数
        print('self.__value:', self.__value)
        print('Root.__total:', Root.__total)

    @classmethod                #修饰器，声明类方法
    def classShowTotal(cls):  #类方法，一般以 cls 作为第一个参数
        print(cls.__total)

    @staticmethod               #修饰器，声明静态方法
    def staticShowTotal():    #静态方法，可以没有参数
        print(Root.__total)

>>> r = Root(3)
>>> r.classShowTotal()         #通过对象来调用类方法
1
>>> r.staticShowTotal()        #通过对象来调用静态方法
1
>>> rr = Root(5)
>>> Root.classShowTotal()      #通过类名调用类方法
2
>>> Root.staticShowTotal()   #通过类名调用静态方法
2
>>> Root.show()                #试图通过类名直接调用实例方法，失败
TypeError: unbound method show() must be called with Root instance as
first argument (got nothing instead)
>>> Root.show(r)               #可以通过这种方法来调用方法并访问实例成员
self.__value: 3
Root.__total: 2
>>> class Test:
    def __init__(self, value):
        self.__value = value

    def __get(self):
        return self.__value

    def __set(self, v):
        self.__value = v

    def __del(self):                        #删除对象的私有数据成员
        del self.__value
```

```
    value = property(__get, __set, __del)   #可读、可写、可删除的属性

    def show(self):
        print(self.__value)

>>> t = Test(3)
>>> t.show()
3
>>> t.value
3
>>> t.value = 5
>>> t.show()
5
>>> t.value
5
>>> del t.value
>>> t.value                              #相应的私有数据成员已删除，访问失败
AttributeError: 'Test' object has no attribute '_Test__value'
>>> t.show()
AttributeError: 'Test' object has no attribute '_Test__value'
>>> t.value =1                           #动态增加属性和对应的私有数据成员
>>> t.show()
1
>>> t.value
1
```

方法	功能说明
__init__()	构造方法，创建对象时自动调用
__del__()	析构方法，释放对象时自动调用
__add__()	+
__sub__()	-
__mul__()	*
__truediv__()	/
__floordiv__()	//
__mod__()	%
__pow__()	**
__eq__()、 __ne__()、 __lt__()、 __le__()、 __gt__()、 __ge__()	==、 !=、 <、 <=、 >、 >=
__lshift__()、__rshift__()	<<、>>
__and__()、__or__()、 __invert__()、__xor__()	&、|、 ~、^
__iadd__()、__isub__()	+=、-=，很多其他运算符也有与之对应的复合赋值运算符
__pos__()	一元运算符+，正号
__neg__()	一元运算符-，负号
__contains__ ()	与成员测试运算符 in 对应
__radd__()、__rsub__	反射加法、反射减法，一般与普通加法和减法具有相同的功能，但操作数的位置或顺序相反很多其他运算符也有与之对应的反射运算符
__abs__()	与内置函数 abs()对应
__divmod__()	与内置函数 divmod()对应

__len__()	与内置函数 len()对应
__reversed__()	与内置函数 reversed()对应
__round__()	对内置函数 round()对应
__str__()	与内置函数 str()对应，要求该方法必须返回 str 类型的数据
__getitem__()	按照索引获取值
__setitem__()	按照索引赋值

第 二 节 课

例 6-2 设计自定义双端队列类，模拟入队、出队等基本操作。

```
class myDeque:
    #构造方法，默认队列大小为10
    def __init__(self, iterable=None, maxlen=10):
        if iterable==None:
            #如果没有提供初始数据，就创建一个空队列
            self._content = []
            self._current = 0
        else:
            #使用给定的数据初始化双端队列
            #_content用于存储实际数据
            #_current表示队列中元素的个数
            self._content = list(iterable)
            self._current = len(iterable)
        #_size表示队列大小
        self._size = maxlen
        if self._size < self._current:
            self._size = self._current

    #析构方法
    def __del__(self):
        del self._content

    #修改队列大小
    def setSize(self, size):
        if size < self._current:
            #如果缩小队列，需要同时删除后面的元素
            for i in range(size, self._current)[::-1]:
                del self._content[i]
            #因为删除了部分元素，所以需要修改队列中元素数量
            self._current = size
        #设置队列大小
        self._size = size

    #在右侧入队
    def appendRight(self, v):
        if self._current < self._size:
            self._content.append(v)
```

```
            self._current = self._current + 1
        else:
            #如果队列已满，则给出提示，并忽略该操作
            print('The queue is full')

    #在左侧入队
    def appendLeft(self, v):
        if self._current < self._size:
            self._content.insert(0, v)
            self._current = self._current + 1
        else:
            print('The queue is full')

    #在左侧出队
    def popLeft(self):
        if self._content:
            self._current = self._current - 1
            return self._content.pop(0)
        else:
            #如果队列是空的，则给出提示，并忽略该操作
            print('The queue is empty')

    #在右侧出队
    def popRight(self):
        #列表中如果有元素则等价于 True，空列表等价于 False
        if self._content:
            self._current = self._current - 1
            return self._content.pop()
        else:
            print('The queue is empty')

    #循环移位
    def rotate(self, k):
        if abs(k) > self._current:
            print('k must <= '+str(self._current))
            return
        self._content = self._content[-k:] + self._content[:-k]

    #元素翻转
    def reverse(self):
        #反向切片，这里也可以调用列表的 reverse()方法
        self._content = self._content[::-1]

    #显示当前队列中元素个数
    def __len__(self):
        return self._current

    #使用 print()打印对象时，显示当前队列中的元素
    def __str__(self):
        return 'myDeque(' + str(self._content)\
```

```
                + ', maxlen='+ str(self._size) + ')'

    #直接对象名当作表达式时，显示当前队列中的元素
    __repr__ = __str__

    #队列置空
    def clear(self):
        self._content = []
        self._current = 0

    #测试队列是否为空
    def isEmpty(self):
        return not self._content

    #测试队列是否已满
    def isFull(self):
        return self._current == self._size

if __name__ == '__main__':
    print('Please use me as a module.')
```

例 6-3 设计自定义栈类，模拟入栈、出栈、判断栈是否为空、是否已满以及改变栈大小等操作。

```
class Stack:
    #构造方法
    def __init__(self, maxlen = 10):
        self._content = []
        self._size = maxlen
        self._current = 0

    #析构方法，释放列表控件
    def __del__(self):
        del self._content

    #清空栈中的元素
    def clear(self):
        self._content = []
        self._current = 0

    #测试栈是否为空
    def isEmpty(self):
        return not self._content

    #修改栈的大小
    def setSize(self, size):
        #不允许新的大小小于已有元素数量
        if size < self._current:
            print('new size must >=' + str(self._current))
            return
```

```
        self._size = size

    #测试栈是否已满
    def isFull(self):
        return self._current == self._size

    #入栈
    def push(self, v):
        if self._current < self._size:
            #在列表尾部追加元素
            self._content.append(v)
            #栈中元素个数加1
            self._current = self._current + 1
        else:
            print('Stack Full!')

    #出栈
    def pop(self):
        if self._content:
            #栈中元素个数减1
            self._current = self._current - 1
            #弹出并返回列表尾部元素
            return self._content.pop()
        else:
            print('Stack is empty!')

    def __str__(self):
        return 'Stack(' + str(self._content)\
               +', maxlen=' + str(self._size) + ')'

    #复用__str__方法的代码
    __repr__ = __str__
```

例 6-4 自定义三维向量类。

```
class Vector3:
    #构造方法，初始化，定义向量坐标
    def __init__(self, x, y, z):
        self.__x = x
        self.__y = y
        self.__z = z

    #与两一个向量相加，对应分量相加，返回新向量
    def add(self, anotherPoint):
        x = self.__x + anotherPoint.__x
        y = self.__y + anotherPoint.__y
        z = self.__z + anotherPoint.__z
        return Vector3(x, y, z)

    #减去另一个向量，对应分量相减，返回新向量
```

```
    def sub(self, anotherPoint):
        x = self.__x - anotherPoint.__x
        y = self.__y - anotherPoint.__y
        z = self.__z - anotherPoint.__z
        return Vector3(x, y, z)

    #向量与一个数字相乘，各分量乘以同一个数字，返回新向量
    def mul(self, n):
        x, y, z = self.__x*n, self.__y*n, self.__z*n
        return Vector3(x, y, z)

    #向量除以一个数字，各分量除以同一个数字，返回新向量
    def div(self, n):
        x, y, z = self.__x/n, self.__y/n, self.__z/n
        return Vector3(x, y, z)

    #查看向量各分量值
    def show(self):
        print('X:{0}, Y:{1}, Z:{2}'.format(self.__x,
                                           self.__y,
                                           self.__z))

    #查看向量长度，所有分量平方和的平方根
    @property
    def length(self):
        return (self.__x**2 + self.__y**2 + self.__z**2)**0.5

#用法演示
v = Vector3(3, 4, 5)
v1 = v.mul(3)
v1.show()
v2 = v1.add(v)
v2.show()
print(v2.length)
```