杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

第7章 面向对象的程序设计

7.1 类和对象的概念

7.2 类和对象的创建

7.3 使用对象编写程序

7.4 封装

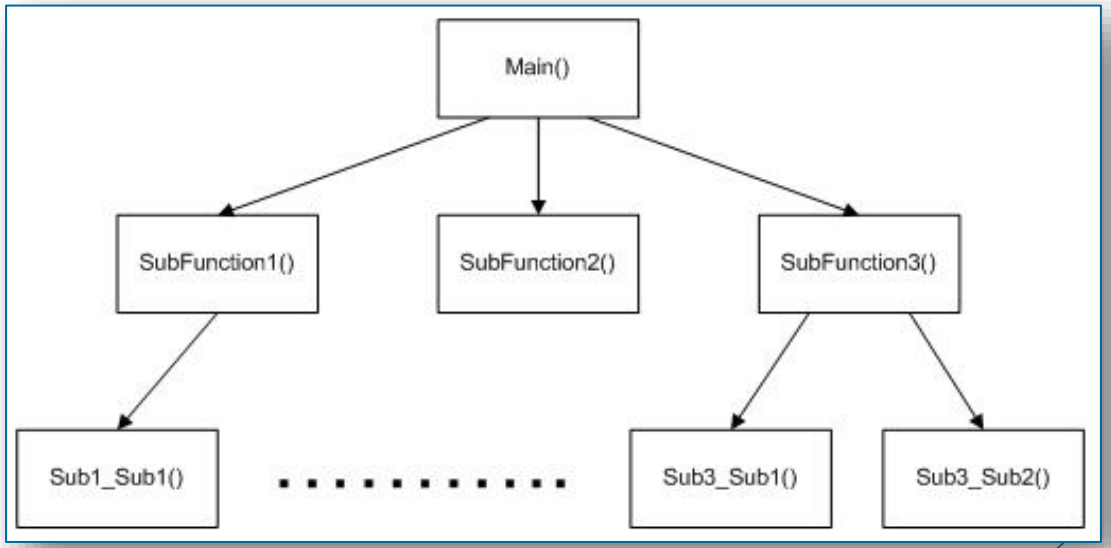
7.5 继承和多态

- 理解类与对象、属性与方法等基本概念
- 能够创建类与子类、以及对象
- 初步理解Python面向对象的特征

面向对象的概念

❖ 程序设计思想——面向过程的程序设计

采用自顶向下的方法，分析出解决问题所需要的步骤，将程序分解为若干个功能模块，每个功能模块用函数来实现。计算机程序即一组函数的顺序执行。



面向对象的概念

❖ 程序设计思想——面向对象的程序设计

使用对象进行程序设计，实现代码重用和设计重用，简化程序开发。先声明事物和情景的类，并基于这些类来创建对象，使用对象来编写程序。创建对象的目的不是为了完成某个步骤，而是为了描述某个事物在解决整个问题中的行为。

在Python中，所有数据类型都可以视为对象，当然也可以自定义对象。

面向过程 vs. 面向对象

【例】处理学生的成绩表，打印学生成绩。

```
#面向过程的程序设计实现
std1 = {'name': 'Michael', 'score': 98 }
std2 = {'name': 'Bob', 'score': 81 }
def print_score(std):
    print('%s: %s' % (std['name'], std['score']))

print_score(std1)
print_score(std2)
```

```
class Student(object):  #定义一类对象
    def __init__(self, name, score):
        self.name = name
        self.score = score
    def print_score(self):
        print('%s: %s' % (self.name, self.score))
```

面向对象的程序设计思想，Student应该被视为一类对象，这类对象拥有name和score这两个属性。要打印一个学生的成绩：

- 首先须创建这个学生对象（包括name和score）
- 给对象发消息，让该对象自己打印出自己的数据（score）。

```
bart = Student('Bart', 59)  #创建两个学生实例
lisa = Student('Lisa', 87)
bart.print_score()  #给对象发消息，即：调用对象的方法
lisa.print_score()
```

7.1 类和对象的概念

❖ 对象（object)

表示现实世界中的一个实体，每个对象都有自己独特的标识、属性和行为。如：一个学生、一张桌子。

- 属性（attribute）：指那些具有当前值的数据域。

如：学生对象具有一个数据域 name

- 行为（behavior）：是由方法（属性的操作函数）定义的。调用对象的一个方法就是要求对象完成一个动作。

如：学生对象可以调用print_score()打印出学生的成绩。

```
lisa.print_score()
```

7.1 类和对象的概念

❖ 类（class）

类是对具有相同属性和行为的同一类对象的描述，它定义了属性（数据）和行为（方法）的模板。

Python类是封装了变量和方法的复合数据类型，其使用一个通用类来定义同一类型的对象，用来定义对象的属性是什么、方法是做什么的。

对象是类的一个实例。

如：`lisa = Student('Lisa', 87)`

7.1 类和对象的概念

- Python的所有类型都是类，包括内置int、str等。
- 字符串、列表、字典、元组等内置数据类型，都具有和类完全相似的语法和用法。

```
>>> help(list)
Help on class list in module builtins:

class list(object)
 |  list() -> new empty list
 |  list(iterable) -> new list initialized fro
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
```

7.2 类和对象的创建

❖ 定义类

使用class关键字来定义类，类的内部实现被列在一个缩进块中。一般格式如下：

```
class ClassName:
    initializer
    methods
```

类中的函数称为方法（methods），包括初始化方法、其他方法。初始化方法总是被命名为__init__，每当类创建新的实例，Python都会自动运行它。

```
class Student:    #定义学生类：包含成员变量和成员方法
    def __init__(self, name, number): #初始化方法
        self.name = name              #成员变量
        self.number = number          #成员变量

    def getInfo(self):                #成员方法
        print(self.name, self.number)
```

一般格式如下：

```
class ClassName:
    initializer
    methods
```

类中的函数称为方法（methods），包括初始化方法、其他方法。初始化方法总是被命名为__init__，每当类创建新的实例，Python都会自动运行它。

7.2 类

```
class Student:    #定义学生类：包含成员变量和成员方法
    def __init__(self, name, number): #初始化方法
        self.name = name                #成员变量
        self.number = number            #成员变量
    def getInfo(self):                    #成员方法
        print(self.name, self.number)
```

❖ self参数：

- 类的所有实例方法都必须有一个名为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参数代表将来要创建的对象本身。
- 在类的方法中访问对象变量（数据成员）时，需要以self为前缀。
- 但在外部通过对象调用对象方法时，并不需要传递这个self参数。

```
Lisa = Student('Lisa', '19021401')
Lisa.getInfo()  #无需传递self参数
```

7.2 类和对

开头和结果各两个下划线

```
class Student:    #定义学生类：包含成员变量和成员方法
    def __init__(self, name, number): #初始化方法
        self.name = name              #成员变量
        self.number = number          #成员变量
    def getInfo(self):                #成员方法
        print(self.name, self.number)
```

```
Lisa = Student('Lisa', '19021401')
Lisa.getInfo()      #无需传递self参数
```

❖ 方法__init__()

- 是一个特殊的方法，每当根据Student类创建新对象时，Python都会自动运行它。
- 形参 self 必不可少，且必须位于其它形参之前。Python调用 __init__( ) 方法来创建对象时，将自动传入实参，self 即指向该对象，我们只需给后两个形参 name 和 number 提供值即可。

7.2 类和对象的创建

❖ 创建对象

- 定义了类之后，就可以创建对象了。一般格式为：

  对象名 = ClassName（实参列表）

- 内置方法 isinstance( )：　用来测试一个对象是否为某个类的实例。

```
>>> s1 = Student("Lisa", "19104101")
>>> isinstance(s1, Student)
True
>>> isinstance(s1, str)
False
>>> type(s1)
<class '__main__.Student'>
```

7.2 类和对象的创建

❖ 访问对象成员

创建对象后，可以使用“.”运算符，访问对象的数据成员或方法成员，一般格式为：

对象名 . 成员

访问数据

```
>>> s1 = Student("Lisa", "19104101")
>>> print("我的名字是" + s1.name + ", 我的学号为" + s1.number)
我的名字是Lisa, 我的学号为19104101
>>> s1.getInfo()
Lisa 19104101
```

调用方法

7.2 类和对象的创建

❖ 属性值

```
>>> s2 = Student("Wu", "1234567")
>>> s2.score
0
```

◆ 给属性指定默认值

可以对某个属性设置默认值，__init__( )方法中无须包含对应的形参。

```
class Student:
    def __init__(self, name, number):
        self.name = name
        self.number = number
        self.score = 0    #为score属性设置默认值

    def getInfo(self):
        print(self.name, self.number)

    def setScore(self, score):    #增加成员方法，用于设置score属性
        self.score = score
```

7.2 类和对象的创建

❖ 属性值

◆ 修改属性的值

```
>>> s2 = Student("Wu", "1234567")
>>> s2.score
0
>>> s2.setScore(90)
>>> s2.score
90
```

1）直接通过对象进行修改

2）通过方法进行设置（推荐）

将值传递给一个方法，避免直接访问属性。

7.3 使用对象编写程序

【例7-1】创建Dog类，并实例化对象Bob。

所有的dog都具有的特性：

- 属性：name、month_age、kind
- 方法（能力）：bark()

构造方法，创建对象时自动调用

```
class Dog:
    def __init__(self,name,kind,month_age):
        self.name = name
        self.month_age = month_age
        self.kind = kind

    def info(self):
        return '狗名: %s (%s,%d个月)' %(self.name,\
                                        self.kind,self.month_age)

    def bark(self):
        print('汪汪')


Bob = Dog('Bob','金毛',9)
print(Bob.info())
Bob.bark()
```

7.4 封装

- **封装是面向对象的主要特性**

所谓封装，就是把客观事物抽象并封装成对象，即：将数据成员、方法等集合在一个整体内。通过访问控制，还可以隐藏内部成员，只允许可信的对象访问或操作自己的部分数据或方法。

将类的实现和类的使用分离，对用户而言，无需知道类时如何实现的，类的实现细节被隐藏，故称为“封装”。

7.4 封装

❖ 类成员

Python类中的成员分为：

- 数据成员（变量、属性）
  - 类数据成员（类变量、类属性）——属于整个类
  - 实例数据成员（实例变量、实例属性）——属于特定的实例，在构造方法__init__( )中定义的，定义与使用时，必须以self作为前缀。
- 方法成员（函数）

7.4

❖ 类属性与

- ◆ 类属性
- ◆ 实例属

必须以

实例属性

类，可以通过

```
class Car:
    price = 100000   #定义类属性price

    def __init__ (self, name):
        self.name = name  #定义实例属性name
        self.color = ''   #定义实例属性color

    def setColor(self, color):   #设置汽车的颜色
        self.color = color

car1 = Car("奥迪")
car2 = Car("宝马")
print(car1.name, Car.price)
Car.price = 310000          #修改类属性
car1.setColor('Blue')       #调用方法修改实例属性
car1.name = '奥迪A6'        #直接修改实例属性
print(car1.name, Car.price, car1.color)
print(car2.name, car2.price, car2.color)
```

```
class Car:
    price = 100000    #定义类属性price

    def __init__ (self, name):
        self.name = name   #定义实例属性name
        self.color = ''    #默认值的实例属性color

    def setColor(self, color):    #定义实例方法setColor()
        self.color = color

    @classmethod    #用来声明类方法
    def getPrice(cls):    #定义类方法getPrice()
        print(cls.price)  #访问类属性
```

```
>>> car3 = Car('吉利')
>>> car3.setColor('white')
>>> print(car3.name, car3.color)
吉利 white
```

```
>>> car4 = Car('沃尔沃')
>>> car4.getPrice()
100000
>>> Car.price = 310000
>>> car4.getPrice()
310000
```

7.4 封装

❖ 私有成员与公有成员

一些常用约定：

- _xxx：受保护成员
- __xxx__：系统定义的特殊成员
- __xxx：私有成员，只有类内自己能访问，不能使用实例直接访问到这个成员。

封装性原则要求：不直接访问类中的数据成员。可以通过定义私有属性，然后定义相应的访问该私有属性的方法（用@property装饰器装饰）。

7.4 封装

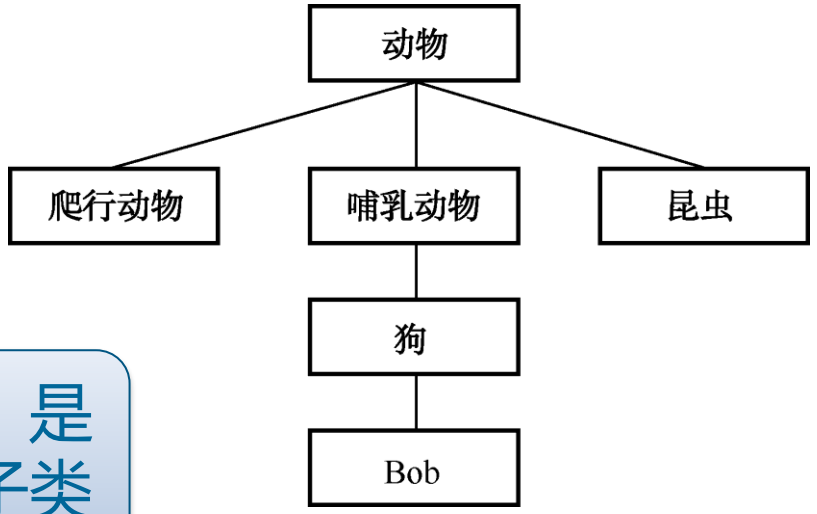
❖ 私有成员与公有成员

```
class Personl:
    def __init__ (self, name, age):
        self.__name = name   #私有属性__name
        self.__age = age     #私有属性__age

    @property
    def name(self):
        return self.__name     #私有属性只能通过name()方法访问
```

```
>>> p = Personl('Li ming', 20)
>>> print(p.__name)
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    print(p.__name)
AttributeError: 'Personl' object has no attribute '__name'
>>> print(p.name)
Li ming
>>>
```

7.5 继承和多态



动物的分类与实例

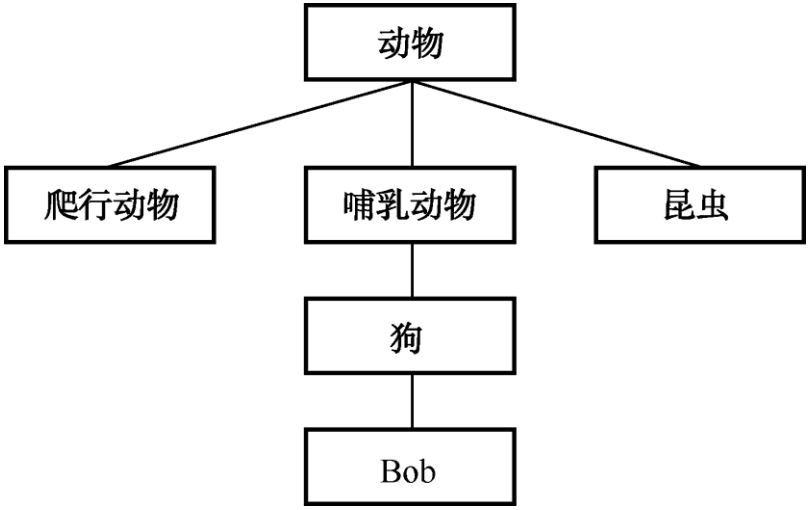
```
class Animals:
    pass
class Mammals(Animals):
    pass
class Dog(Mammals):
    pass
```

“哺乳动物”是“动物”的子类

父类

子类Mammals可以继承父类Animals的所有属性，同样，子类Dog也可以继承父类mammals的所有属性。

7.5 继承和多态



❖ 继承

当已经存在一个类，需要另外再创建一个和已有类非常相似的类时，通常不必将同一段代码重复多次，而是用继承。在类上添加关联，使得位于下层的类可以“继承”位于关系上层的类的属性。继承有利于代码的复用和规模化。和其他语言不同的是，Python中的类还具有多继承的特性，即一个类可以有多个父类。

【例】 继承一个父类。

当调用check()方法时，Company本身没有check方法，代码会向上自动检测父类Scale中是否存在check方法。

```
class Scale:
    def check(self):
        if self.count_person > 500:
            print ("%s是个大公司." %self.name)
        else:
            print ("%s是个小公司." %self.name)

class Company(Scale):
    def __init__(self, name, count):
        self.name = name
        self.count_person = count

if __name__ == "__main__":
    my_company = Company("ABC", 800)
    my_company.check()
```

只有一个父类Scale：单继承

【例】 继承多个父类。

```
class Scale:
    def check(self):
        if self.count_person > 500:
            return "%s是个大公司." %self.name
        else:
            return "%s是个小公司." %self.name
class Detail:
    def show(self, scale):
        print("%s 公司有%s名员工."%(scale, self.count_person))

class Company(Scale, Detail):
    def __init__(self, name, count):
        self.name = name
        self.count_person = count

if __name__ == "__main__":
    my_company = Company("ABC", 800)
    company_scale = my_company.check()
    my_company.show(company_scale)
```

多继承

Company分别继承了类Scale和类Detail，可以调用父类中的check()方法和show()方法。

7.5 继承和多态

❖ 多态

多态即多种状态，是指在事先不知道对象类型的情况下，可以自动根据对象的不同类型，执行相应的操作。

很多内建运算符以及函数、方法都能体现多态的性质。例如“+”运算符，在连接数值类型变量时表示加法操作，在连接字符串时则表示拼接。

```
>>> 2+3
5
>>> '123'+'45'
'12345'
>>> [1,2,3]+[4,5]
[1, 2, 3, 4, 5]
```

```
>>> 2*3
6
>>> 'Hello'*3
'HelloHelloHello'
>>> [1,2,3]*3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

7.5 继承和多态

❖ 多态

多态即多种状态，是
，可以自动根据对象的不

【例】 函数的多态性举例

函数repr()返回一个对象的可打印字符串，无须事先知道对象是什么类型，该函数也表现了Python多态特性。

```
>>> def length(x):
    print( repr(x) ,"的长度为", len(x))
>>> length('aaa')
'aaa'的长度为3
>>> length([1,2,3,4,5])
[1, 2, 3, 4, 5]的长度为5
```