Python 语言程序设计

陈 峦 副教授

13880209111，chluan@uestc.edu.cn

研究院大楼316#

第九章
面向对象程序设计

- 传统的面向过程程序设计是基于问题的求解过程来组织程序流程的。
- 面向对象程序设计（**Object-Oriented programming，OOP**）则以对象作为程序的主体，将程序和数据封装于其中，以提高软件的重用性、灵活性和扩展性。

- 面向对象程序设计是按照人们认识客观世界的系统思维方式，采用基于对象的概念建立问题模型，模拟客观世界，分析、设计和实现软件的办法。

9.1 面向对象程序设计概述

- 面向对象语言的三大核心内容是封装（类和对象）、继承（派生）和多态。

1. 类和对象

- 对象的特征用数据（变量）来表示，称为属性（property）。
- 对象的行为用程序代码来实现，称为对象的方法(method)。
- 任何对象都是由属性和方法组成的。
- 属性：特征，状态，数据，变量
- 方法：行为，服务，功能，运算操作，函数

- 对象是类的实例，类定义了属于该类的所有对象的共同特性（属性和行为）。
- 类（class）是具有相同属性和行为的一组对象的集合，它为属于该类的全部对象提供了统一的抽象描述。
- 任何对象都是某个类的实例（instance）。

- Python的面向对象与其他程序设计语言（如C++、Java）的面向对象有一些差异。
- 在Python中，一切都是对象，类本身是一个对象（类对象），类的实例也是对象。
- Python中的变量、函数都是对象。

2. 继承和派生

- 继承（inheritance）反映的是类与类之间抽象级别的不同。
- 父类表现出的是共性和一般性，子类表现出的是个性和特性，父类的抽象级别高于子类。
- 通过继承，可以实现以类为单位的代码复用。

3. 多态

- 多态（polymorphism）是指同一名字的方法产生了多个不同的动作行为，即不同的对象收到相同的消息时产生不同的行为方式。
- 一个名字，多种语义；相同界面，多种实现。
- 重载：函数重载，运算符重载，同名覆盖等。
- Python是解释性语言，只有运行时多态。

4. 从面向过程到面向对象

- 假设要处理学生的成绩表，为了表示一个学生的成绩，面向过程的程序可以用一个字典表示。
- 例：面向过程程序

```
def show(s):
    print("{}:{}".format(s["name"],s["score"]))
    return
def main():
    s1={"name":"张一","score":95}
    s2={"name":"张二","score":88}
    t={}
    for i in range(1):
        t["name"]=input("name=")
        t["score"]=input("score=")
    show(s1)
    show (s2)
    show (t)
    return
main()
```

- 如果采用面向对象的程序设计思想，首先考虑的不是程序的执行流程，而是如何表达学生的信息：
- 学生（Student）这种数据类型应该被视为一个类，这个类拥有name和score这两个属性（property）。
- 如果要打印一个学生的成绩，首先必须创建出这个学生对应的对象，然后，给对象发一个输出成绩的消息，让对象把数据打印出来。

```
class Student(object):  #继承父类
    def __init__(s,name,score):  #构造方法
        s.name=name  #实例属性
        s.score=score   #实例属性
        return
    def show(s):   #实例方法
        print("{}:{}".format(s.name,s.score))
        return
def main():
    s1=Student("张一",95)  #实例对象
    s2=Student("张二",88)  #实例对象
    s1.show()
    s2.show()
    return
main()
```

9.2 类与对象

- 类是一种抽象数据类型，对象是具有这种数据类型的变量。
- 类是抽象的，不占用内存空间；而对象是具体的，占用存储空间。
- 当创建对象之后，系统将为该对象分配内存空间。

9.2.1 类的定义

- 类是一种广义的数据类型，这种数据类型中的元素（或成员）既包含数据，也包含操作数据的函数。
- 定义类的一般格式：
- class 类名:
- 　类体

- 当定义一个类时，会自动生成一个全局的类对象，其名字与类名相同。（这一点和C++中的类不同，C++的类名只代表数据类型）
- 类对象支持两种操作：引用和实例化。
- 引用：通过类对象去访问类中的类属性或类方法；
- 实例化：创建一个类对象的实例，称为实例对象。
- 注意区别：类、类对象、实例对象

- 例：
- class A:　#定义类
- 　x=2　　#定义类属性
- def main():
- 　print("1:A.x=",A.x) #通过类对象访问类属性
- 　A.x=3　#通过类对象改变类属性
- 　print("2:A.x=",A.x)
- 　return
- main()

```
1:A.x= 2
2:A.x= 3
```

- 类对象的名字与类名相同（大小写也必须相同）。
- 类对象不需要单独声明（定义）。当一个类定义好之后，类对象就由系统自动创建好了，直接使用即可。

9.2.2 对象的创建和使用

- 在Python中，用赋值的方式创建类的实例，一般格式为：
- 实例对象名=类名(参数列表)
- 这种方式创建的对象称为实例对象。
- 在一个类中，类对象只有一个，但实例对象可以有多个。
- 实例对象名的首字母通常小写（习惯，非语法要求）。

- 创建对象后，可以通过实例对象来访问这个类的属性和方法（函数），一般格式为：
- 对象名.属性名
- 对象名.函数名()
- “.”运算符：成员运算符

- 例：
- class A:
- 　x=2　#定义类属性
- def main():
- 　print("A.x=",A.x) #通过类对象访问类属性
- 　a=A()　#创建实例对象
- 　print("a.x=",a.x) #通过实例对象访问类属性
- 　return
- main()

```
A. x= 2
a. x= 2
```

- 例：
- class A:
- 　x=2
- 　def show(s): #实例方法
- 　　print("s.x=",s.x) #通过实例对象访问类属性
- 　　return
- def main():
- 　print("A.x=",A.x)
- 　a=A()
- 　a.show()　#通过实例对象调用实例方法
- 　print("a.x=",a.x)
- 　return
- main()

9.3 属性和方法

- 在类中定义的函数称为方法。
- 在类外定义的函数称为函数。
- 注意区别：属性、方法、函数

- 例：
- def f(x):　　#全局函数
- 　print("f:x=",x)
- class A:
- 　y=2;　　#类属性
- 　def g(self,z):　　#实例方法
- 　　print("g:y=",self.y,",z=",z)
- def main():
- 　m=3　　#局部变量
- 　f(m)　　#调用全局函数
- 　a=A()
- 　a.g(m)　　#通过实例对象调用实例方法
- main()

9.3.1 属性和方法的访问控制

1. 属性的访问控制

- 例：
- class A:
- 　　x=2　　#类属性
- def main():
- 　　a=A()
- 　　print(A.x,a.x)
- main()

- 一般类的属性默认是公有的，公有属性可以直接在类外通过对象名访问。
- 如果想将类的属性定义为私有的，则需在属性名前面加2个下画线“__”。
- 类的私有属性是不能够在类外通过对象名来进行访问的。
- 在类外访问类中的私有属性将出错。

- 例：
- class A:
- 　__x=2　　#私有类属性
- def main():
- 　a=A()
- 　#print(A.__x)　　#非法
- 　#print(a.__x)　　#非法
- main()
- #AttributeError: 'A' object has no attribute '__x'

- 例：
- class A:
- 　__x=2　　#私有的类属性
- 　def f(self):　　#实例方法
- 　　print("f:self.__x=",self.__x)　#合法
- 　　print("f:A.__x=",A.__x)　　#合法
- def main():
- 　a=A()
- 　a.f()
- 　#print("main:a.__x=",a.__x)　　#非法
- 　#print("main:A.__x=",A.__x)　　#非法
- main()

- 例：
- class A:
- 　__x=2　#私有的类属性
- def main():
- 　a=A()
- 　#print(a.__x)　　#非法
- 　a.__x=3　　#定义实例属性
- 　print(a.__x)　#合法
- 　del a.__x　　#删除实例属性
- 　#print(a.__x)　#非法
- main()

- 如果类中的方法在方法名前面加了2个下画线“__”，则表示该方法是私有的，否则为公有的。

- 例：
- class A:
- 　def f(self):　　#公有的实例方法
- 　　print("AA")
- 　def __f(self):　#私有的实例方法
- 　　print("BB")
- def main():
- 　a=A()
- 　a.f()　　#合法
- 　#a.__f()　#非法
- 　#原因：全局函数中不能访问类中的私有实例方法
- main()

2. 方法的访问控制

- 类中的实例方法至少应有一个变量参数，一般命名为“self”(习惯，非语法要求)，而且该参数必须作为形参表的第一个参数，即必须放于形式参数表的最左边。
- 该变量实质是当前实例对象。（类似于C++中的*this、Java中的this）

- 程序中self是当前对象自身的意思，在用某个对象调用该方法时，就将该对象作为第一个参数传递给self。

- 例：
- class A:
- 　　x=2
- 　　y=3
- 　　def show(self,t=4):
- 　　　　print(self.x,self.y,A.x,A.y,t)
- 　　　　return
- a=A()
- a.show()

2 3 2 3 4

- 例：
- class A:
- 　x=2　　#公有的类属性
- 　__y=3　　#私有的类属性
- 　def show(this,t):　#公有的实例方法
- 　　print(this.x,this.__y,t)
- 　　return
- a=A()
- a.show(4)　　2 3 4

- 实例方法只能用实例对象来调用，不能用类对象来调用实例方法。
- 同理，实例属性也只能用实例对象来访问，不能用类对象来访问实例属性。
- 注意：类属性即可用类对象访问，也可用实例对象访问。

- 例：
- class A:
- 　　def f(self):　　#公有的实例方法
- 　　　　print("AA")
- def main():
- 　　a=A()
- 　　a.f()　　#合法
- 　　#A.f()　　#非法
- main()

9.3.2 类属性和实例属性

1. 类属性

- 类属性（class attribute）是类的属性，它被所有类对象和实例对象共有，在内存中只存在一个副本。
- （与C++的静态成员变量、Java的类变量类似）
- 公有的类属性，在类外可以通过类对象和实例对象访问。

- 类属性是在类中方法之外定义的，它属于类，可以通过类对象访问。
- 也可以通过实例对象来访问类属性，但不提倡这样做（容易出错，可能会造成类属性值不一致）。

- 类属性还可以在类定义结束之后通过类名（类对象）增加。
- 实例对象也可以在类定义结束之后通过实例对象名增加实例属性。
- 注意区别：类属性、实例属性

- 例：
- class A:
- 　x=2　　#定义类属性
- def main():
- 　print(A.x)
- 　A.y=3　　#定义类属性
- 　print(A.x,A.y)
- main()

- 例：
- class A:
- 　def f(self):
- 　　self.x=2　　#定义实例属性
- 　　y=3　　#定义局部变量
- 　　print(self.x,y)
- def main():
- 　a=A()　　#定义实例对象
- 　a.f()
- 　a.z=4　　#定义实例属性
- 　print(a.x,a.z)
- main()

```
2 3
2 4
```

```
class A:
    x=2  #定义类属性
    def show(self):
        print(self.x)
        return
def main():
    print("0:A.x=",A.x) #通过类对象访问类属性
    A.x=3               #通过类对象修改类属性
    print("1:A.x=",A.x) #通过类对象访问类属性
    a=A()               #定义实例对象
    print("2:a.x=",a.x)  #通过实例对象访问类属性
    a.x=4               #定义实例属性
    print("3:A.x=",A.x)  #通过类对象访问类属性
    print("4:a.x=",a.x)  #通过实例对象访问实例属性
    return
main()
```

```
0:A.x= 2
1:A.x= 3
2:a.x= 3
3:A.x= 3
4:a.x= 4
```

- 例：
- class A:
- 　x,y=2,3　　#定义类属性
- 　def show(self):
- 　　print(self.x,self.y)
- 　　return
- def main():
- 　a=A()
- 　a.x=5　　#定义实例属性
- 　a.show()
- 　print(A.x,A.y)
- 　print(a.x)
- 　return
- main()

```
5 3
2 3
5
```

- 例：
- class A:
- 　x=2　　#定义类属性
- 　def show(self):
- 　　A.y=3　　#定义类属性
- 　　self.z=4　　#定义实例属性
- 　　print(self.x,self.y,self.z)
- def main():
- 　a=A()
- 　a.show()
- 　print(A.x,A.y)
- 　print(a.x,a.y,a.z)
- main()

```
2 3 4
2 3
2 3 4
```

- 例：
- class A:
- 　x=2　　#定义类属性
- 　def show(self):
- 　　self.x=3　#定义实例属性
- 　　A.y=4　　#定义类属性
- 　　print(A.x,A.y)
- def main():
- 　a=A()
- 　a.y=5　　#定义实例属性
- 　a.show()
- 　print(A.x,A.y)
- 　print(a.x,a.y)
- main()

```
2 4
2 4
3 5
```

- 例：
- class A:
- 　x=2　　#定义类属性
- 　def show(self):
- 　　A.y=4　　#定义类属性
- 　　a.y=5　　#定义实例属性
- 　　print(a.x,a.y)
- 　　self.x=3　　#定义实例属性
- a=A()
- a.show()
- print(A.x,A.y)
- print(a.x,a.y)

```
2 5
2 4
3 5
```

- 例：
- class A:
- x=2
- a=A()
- b=A()
- print(a.x,b.x,A.x)
- a.x=3
- print(a.x,b.x,A.x)
- b.x=4
- print(a.x,b.x,A.x)

```
2 2 2
3 2 2
3 4 2
```

- 例：
- class A:
- x=2
- def f(s):
- s.x=3
- a=A()
- b=A()
- a.f()
- print(a.x,b.x,A.x)
- a.x=4
- print(a.x,b.x,A.x)
- b.x=5
- print(a.x,b.x,A.x)

```
3 2 2
4 2 2
4 5 2
```

- **2. 实例属性**
- 实例属性（instance attribute）不需要在类中显式定义，通常是在构造方法__init__中定义的，定义时以self作为前缀。
- 在其他方法中也可以定义实例属性，但不提倡这么做（分散定义，太乱，可读性差）。
- 实例属性属于该实例对象，只能通过该实例对象访问。

- 例：
- class A:
- 　　def __init__(self,m,n):
- 　　　　self.x=m　　#公有的实例属性
- 　　　　self.__y=n　#私有的实例属性
- a=A(2,3)
- print(a.x)
- #print(a.__y)　　　#非法
- #AttributeError: 'A' object has no attribute '__y'

```
class A:
    def __init__(self,m,n):
        self.x=m     #公有的实例属性
        self.__y=n   #私有的实例属性
    def f(s):
        s.z=4        #公有的实例属性
        print(s.x,s.__y,s.z)   #合法
a=A(2,3)
print(a.x)
#print(a.__y)   #非法
a.f()
print(a.x,a.z)
```

```
2
2 3 4
2 4
```

- 例：
- class A:
- 　　def __init__(self,m,n):
- 　　　　self.x=m
- 　　　　self.y=n
- 　　def show(self):
- 　　　　print(self.x,self.y)
- a=A(2,3)
- a.show()
- b=A(4,5)
- b.show()

```
2 3
4 5
```

- 如果需要在类外修改类属性，则必须先通过类对象去引用，然后进行修改。
- 如果通过实例对象去引用，则会产生一个同名的实例属性，这种方式修改的是实例属性，不会影响到类属性，并且之后如果通过实例对象去引用该名称的属性，实例属性会强制屏蔽掉类属性，即引用的是实例属性，除非删除了该实例属性。

- 例：
- class A:
- x=2
- a=A()
- print(a.x)
- A.x=3
- print(a.x)
- a.x=4
- print(a.x,A.x)

```
2
3
4 3
```

- 例：
- class A:
- x=2
- print("1:A.x=",A.x)
- a=A()
- print("2:a.x=",a.x)
- a.x=3
- print("3:A.x=",A.x)
- print("4:a.x=",a.x)
- del a.x
- print("5:A.x=",A.x)
- print("6:a.x=",a.x)

```
1:A.x= 2
2:a.x= 2
3:A.x= 2
4:a.x= 3
5:A.x= 2
6:a.x= 2
```

9.3.3 类的方法

1. 类中内置的方法

- 在Python类中有一些内置方法，其方法名以两个下画线开始和以两个下画线结束。
- 类中最常用的内置方法就是构造方法（__init__）和析构方法（__del__）。

（1）构造方法

- 构造方法__init__(self,……)在创建实例对象时自动调用，用于初始化属性和为实例对象分配资源。
- 如果在类中用户自己没有重新定义构造方法，系统就会自动生成一个默认的构造方法。

- 例：
- class Person:
- 	def __init__(self,s):
- 		self.name=s
- 	def sayHi(self):
- 		print('Hello,my name is',self.name)
- p=Person('AA')
- p.sayHi()

例：

```
class A:
    def __init__(self,m,n):
        self.x=m
        self.y=n
    def __init__(self):
        self.x=2
        self.y=3
    def show(self):
        print(self.x,self.y)
a=A()
a.show()
#b=A(4,5)
#TypeError: __init__() takes 1 positional argument but 3 were given
```

例：

```
class A:
    def __init__(self):
        self.x=2
        self.y=3
    def __init__(self,m,n):
        self.x=m
        self.y=n
    def show(self):
        print(self.x,self.y)
a=A(4,5)
a.show()
#b=A()
#TypeError: __init__() missing 2 required positional arguments: 'm' and 'n'
```

结论：

- 如果用户在类中声明（定义）多个名字相同但形式参数不同的构造方法，Python会使用类中最后一个声明的构造方法，前面定义的则被同名覆盖了。
- 如果想实现类似C++或Java中的方法重载的话，则需要使用变通方法。

- 例：构造方法重载之一

```
class A:
    def __init__(self,x=0,y=0):
        self.x=x
        self.y=y
    def show(self):
        print(self.x,self.y)
```

```
a=A()
b=A(1)
c=A(2,3)
a.show()
b.show()
c.show()
```

例：构造方法重载之二

```
class A:
    def __init__(self,*arg):
        if len(arg)==0:
            self.init_1()
        elif len(arg)==1:
            self.init_2(*arg)
        elif len(arg)==2:
            self.init_3(*arg)
        else:
            print("Wrong arguments!")
    def init_1(self):
        self.x=0
        self.y=0
    def init_2(self,x):
        self.x=x
        self.y=0
    def init_3(self,x,y):
        self.x=x
        self.y=y
    def show(self):
        print(self.x,self.y)
```

```
a=A()
b=A(1)
c=A(2,3)
d=A(4,5,6)
a.show()
b.show()
c.show()
```

- 例：构造方法重载之三

```
class A:
    def __init__(self,*arg):
        if len(arg)==0:
            self.x=0
            self.y=0
        elif len(arg)==1:
            self.x=arg[0]
            self.y=0
        elif len(arg)==2:
            self.x=arg[0]
            self.y=arg[1]
        else:
            print("Wrong arguments!")
    def show(self):
        print(self.x,self.y)
```

```
a=A()
b=A(1)
c=A(2,3)
d=A(4,5,6)
a.show()
b.show()
c.show()
```

- 例：
- class A:
- 　　def __init__(self):
- 　　　　print("AA")
- a=A()
- b,c=A(),A()
- A()
- #只要创建实例对象，就一定要调用构造方法；
- #只要调用了构造方法，就一定创建了实例对象。

（2）析构方法

- 析构方法：__del__(self)
- 在实例对象生成期结束时将自动调用析构函数。
- 析构方法用于释放实例对象占用的系统资源，不需要显式调用。
- 注意区别：类的普通成员方法、构造方法、析构方法

- 例：
- class A:
- 　　def __init__(s):　　#构造方法
- 　　　　print("AA")
- 　　def __del__(s):　　#析构方法
- 　　　　print("aa")
- def f():
- 　　b=A()
- def main():
- 　　a=A()
- 　　del a
- 　　f()
- main()

```
AA
aa
AA
aa
```

2. 类方法、实例方法和静态方法

（1）类方法

- 类方法是类对象所拥有的方法，需要用修饰器“@classmethod”来标识其为类方法。
- 类方法的第一个参数必须是类对象，该参数通常命名为“cls”。（习惯，非语法要求）
- 可以通过实例对象和类对象去访问类方法。
- 类方法可以对类属性进行修改。

- 例：
- class Person:
- 　　place="Chengdu"
- 　　@classmethod
- 　　def getPlace(cls): #类方法
- 　　　　return cls.place
- p=Person()
- print(p.getPlace()) #可以通过实例对象引用
- print(Person.getPlace()) #可以通过类对象引用

- 例：
- class Person:
- 　　place="Chengdu"
- 　　@classmethod
- 　　def getPlace(cls):
- 　　　　return cls.place
- 　　@classmethod
- 　　def setPlace(cls,p):
- 　　　　cls.place=p
- p=Person()
- p.setPlace("Shanghai") #修改类属性
- print(p.getPlace())
- print(Person.getPlace())

- 例：
- class A:
- 　def f(self):　#实例方法
- 　　print("AA")
- 　@classmethod
- 　def g(cls):　#类方法
- 　　print("BB")
- a=A()
- a.f()
- a.g()
- A.g()
- #A.f()　#非法
- #TypeError: f() missing 1 required positional argument: 'self'

- 例：
- class A:
- 　x=2　　#类属性
- 　@classmethod
- 　def set(cls,y):　#类方法
- 　　cls.x=y
- a=A()
- print(a.x,A.x)
- A.set(3)
- print(a.x,A.x)

```
2 2
3 3
```

- 例：
- class A:
- x=2
- def set(self,y): #实例方法
- self.x=y #定义实例属性
- a=A()
- print(a.x,A.x)
- a.set(3)
- print(a.x,A.x)

```
2 2
3 2
```

（2）实例方法

- 实例方法至少有一个实例对象形式参数（通常命名为self），它必须位于形式参数表的最左端，即作为其第一个形式参数。
- 在类外，实例方法只能通过实例对象去调用，不能通过类对象去调用。

- 例：
- class A:
- 　　def f(s):　　#实例方法
- 　　　　print("AA")
- 　　@classmethod
- 　　def g(c):　　#类方法
- 　　　　print("BB")
- a=A()
- a.f()
- a.g()
- #A.f()　　#非法
- A.g()

（3）静态方法

- 静态方法需要通过修饰器“@staticmethod”来进行修饰。
- 静态方法可以没有形式参数（即不需要多定义参数）。
- 静态方法中不需要额外定义参数，如果在静态方法中引用类属性，则必须通过类对象来引用。

- 例：
- class A:
- @staticmethod
- def f(): #静态方法
- print("AA")
- a=A()
- a.f()
- A.f()

- 例：
- class A:
- 　　def f(s):　　#实例方法
- 　　　　print("AA")
- 　　@classmethod
- 　　def g(c):　　#类方法
- 　　　　print("BB")
- 　　@staticmethod
- 　　def h():　　#静态方法
- 　　　　print("CC")
- a=A()
- a.f()
- a.g()
- a.h()

- 例：
- class A:
- 　x=2　　#类变量（类属性）
- 　@staticmethod
- 　def f():
- 　　x=3　　#局部变量
- 　　print(a.x,A.x,x)
- a=A()
- a.x=4　　#实例变量（实例属性）
- a.f()
- A.f()

```
4 2 3
4 2 3
```

- 例：
- class A:
- 　　def f(self):
- 　　　　print("AA")
- 　　@classmethod
- 　　def f(cls):
- 　　　　print("BB")
- 　　@staticmethod
- 　　def f():
- 　　　　print("CC")
- a=A()
- a.f()
- A.f()

- 例：
- class A:
- @classmethod
- def f(cls):
- print("BB")
- @staticmethod
- def f():
- print("CC")
- def f(self):
- print("AA")
- a=A()
- a.f()

- 例：
- class A:
- @staticmethod
- def f():
- print("CC")
- def f(self):
- print("AA")
- @classmethod
- def f(cls):
- print("BB")
- a=A()
- a.f()
- A.f()

- 对于类属性和实例属性，如果在类方法中引用某个属性，则该属性必定是类属性；
- 如果在实例方法中引用某个属性（不做更改），并且存在同名的类属性，此时若实例对象有该名称的实例属性，则实例属性会屏蔽类属性，即引用的是实例属性；（自己有，优先用自己的）
- 若实例对象没有该名称的实例属性，则引用的是类属性。（自己没有，就用“国家”的）

- 如果在实例方法中更改某个属性，并且存在同名的类属性，此时若实例对象有该名称的实例属性，则修改的是实例属性；
- 若实例对象没有该名称的实例属性，则会创建一个同名称的实例属性。
- 想要修改类属性：如果在类外，可以通过类对象修改；如果在类中，只能在类方法中进行修改。

- 从类方法、实例方法以及静态方法的定义形式可以看出，类方法的第一个参数是类对象cls，那么通过cls引用的必定是类对象的属性和方法；
- 实例方法的第一个参数是实例对象self，那么通过self引用的可能是类属性，也有可能是实例属性，在存在相同名称的类属性和实例属性的情况下，实例属性优先级更高。

```
class A:
    x=2    #类属性
    def f(s):
        s.x=3
    @classmethod
    def g(c):
        c.x=4
    @staticmethod
    def h():
        x=5    #局部变量
        A.x=6
        a.x=7
```

```
a=A()
print(a.x,A.x)
a.f()
print(a.x,A.x)
a.g()
print(a.x,A.x)
a.h()
print(a.x,A.x)
```

```
2 2
3 2
3 4
7 6
```

9.4 继承和多态

- 面向对象程序设计方法的三大特征：封装、继承和多态。

9.4.1 继承

- 面向对象程序设计的核心思想是提高代码的重用率。
- 当设计一个新类时，不必从零开始定义，通过继承已有的类，实现以类为单位的代码重用。
- 新类（子类、派生类）从已有的类（父类、超类、基类）那里获得属性和行为（方法），这种现象称为类的继承（inheritance）。

- 通过继承，在定义一个新类时，先把已有类的功能包含进来，然后再定义一些新功能，或对继承来的某些功能重新定义（同名覆盖），从而实现类的重用。
- 从已有类产生新类的过程就称为类的派生（derivation）。（派生是继承的另一种说法）

- 在继承关系中，被继承的类称为父类、超类、基类，继承的类称为子类、派生类。
- 在Python中，类继承的定义格式为：
- class 子类名(父类名):
- 　类体
- 在定义一个类时，在类名后面紧跟一对括号，在括号中指定所继承的父类。
- 如果有多个父类，多个父类名之间用逗号隔开。

- 例：以大学里的学生和教师为例，可以定义一个父类**UniversityMember**，然后类**student**和类**Teacher**分别继承类**UniversityMember**。

```
class UniversityMember:    #定义父类
    def __init__(self,name,age):
        self.name=name
        self.age=age
        print('init UniversityMember:',self.name)
    def tell(self):
        print('name:{};age:{}'.format(self.name,self.age))
```

```
class Student(UniversityMember):  #定义子类Student
    def __init__(self,name,age,marks):
        UniversityMember.__init__(self,name,age)
        self.marks=marks
        print('init Student:',self.name)
    def tell(self):
        UniversityMember.tell(self)
        print('marks:',self.marks)
```

```
class Teacher(UniversityMember):    #定义子类Teacher
    def __init__(self,name,age,salary):
        UniversityMember.__init__(self,name,age)
        #显式调用父类构造方法
        self.salary=salary
        print('init Teacher:',self.name)
    def tell(self):
        UniversityMember.tell(self)
        print('salary:',self.salary)
```

- s=Student('Brenden',18,92)
- t=Teacher('Jasmine',28,2450)
- members=[s,t]
- print
- for member in members:
- member.tell()

- （1）在Python中，如果父类和子类都重新定义了构造方法__init__()，在进行子类实例化的时候，子类的构造方法不会自动调用父类的构造方法，必须在子类中显式调用。
- 同理，子类对象消亡时也不会自动调用父类的析构函数，也必须子类中显式调用。
- （这一点与C++和Java不同）

- 例：
- class A:
- 　　def __init__(self):
- 　　　　print("AA")
- 　　def __del__(self):
- 　　　　print("aa")
- class B(A):
- 　　def __init__(self):
- 　　　　print("BB")
- 　　def __del__(self):
- 　　　　print("bb")
- b=B()
- del b

- 例：
- class A:
- 　　def __init__(self):
- 　　　　print("AA")
- 　　def __del__(self):
- 　　　　print("aa")
- class B(A):
- 　　def __init__(self):
- 　　　　A.__init__(self)
- 　　　　print("BB")
- 　　def __del__(self):
- 　　　　print("bb")
- 　　　　A.__del__(self)
- b=B()
- del b

例：

```
class A:
    def __init__(self):
        print("AA")
    def __del__(self):
        print("aa")
class B(A):
    def __init__(self):
        A.__init__(self)
        print("BB")
    def __del__(self):
        print("bb")
        A.__del__(self)
def main():
    b1=B()
    b2=B()
main()
```

例：

```
class A:
    def __init__(self):
        print("AA")
    def __del__(self):
        print("aa")
class B(A):
    def __init__(self):
        A.__init__(self)
        print("BB")
    def __del__(self):
        print("bb")
        A.__del__(self)
def f():
    b1=B()
f()
b2=B()
```

- 例：
- class A:
- 　　def __init__(self):
- 　　　　print("AA")
- 　　def __del__(self):
- 　　　　print("aa")
- class B(A):
- 　　pass #空类（没有任何属性和方法的类）
- def main():
- 　　b1=B()
- 　　b2=B()
- main()

- 例：
- class A:
- 　def __init__(self,m):
- 　　self.x=m
- class B(A):
- 　def __init__(self,m,n):
- 　　A.__init__(self,m)
- 　　self.y=n
- def main():
- 　b=B(2,3)
- 　print(b.x,b.y)
- main()

例：

```
class A:
    def __init__(self,m):
        self.x=m
    def show(self):
        print("x=",self.x)
class B(A):
    def __init__(self,m,n):
        A.__init__(self,m)
        self.y=n
    def show(self):
        A.show(self)
        print("y=",self.y)
def main():
    b=B(2,3)
    b.show()
main()
```

- （2）在子类中可以用“父类名.方法” 格式调用父类的方法，此时需要传递self参数。
- 子类可以继承父类的公有属性和公有方法，在子类中可以通过父类名来访问它们。
- 子类不能继承父类的私有属性和私有方法，在子类中不能通过父类名访问父类的私有属性和私有方法。

- 例：
- class A:
- 　　def f(self):
- 　　　　print("AA")
- class B(A):
- 　　def g(self):
- 　　　　self.f()
- 　　　　A.f(self)
- 　　　　B.f(self)
- def main():
- 　　b=B()
- 　　b.g()
- main()

```
class A:
    def f(self):
        print("AAff")
    @classmethod
    def g(cls):
        print("AAgg")
class B(A):
    def s(self):
        self.f()
        A.f(self)
        B.f(self)
        self.g()
        A.g()
        B.g()
    @classmethod
    def t(cls):
        cls.g()
        A.g()
        B.g()
```

```
def main():
    b=B()
    b.s()
    B.t()
main()
```

例：

```
class A:
    def f(self):
        print("AA")
    def __f(self):
        print("aa")
class B(A):
    def g(self):
        self.f()
        A.f(self)
        B.f(self)
        #self.__f()  #非法
        #A.__f(self)  #非法
        #B.__f(self)  #非法
def main():
    b=B()
    b.g()
main()
```

例：

```
class A:
    def f(self):
        print("AA")
        self.__f()
    def __f(self):
        print("aa")
class B(A):
    def g(self):
        self.f()
        A.f(self)
        B.f(self)
def main():
    b=B()
    b.g()
main()
```

例：

```
class A:
    def __init__(self):
        print("AA")
class B(A):
    def __init__(self):
        print("BB")
class C(B):
    def __init__(self):
        print("CC")
class D(C):
    pass
def main():
    d=D()
main()
```

例：

```
class A:
    def __init__(self):
        print("AA")
class B(A):
    def __init__(self):
        A.__init__(self)
        print("BB")
class C(B):
    def __init__(self):
        B.__init__(self)
        print("CC")
class D(C):
    pass
def main():
    d=D()
main()
```

9.4.2 多重继承

- 单继承是指子类只有一个直接父类。
- 多重继承（multiple inheritance）是指一个子类有两个或多个直接父类，子类从两个或多个直接父类中继承所需的属性和方法。
- Python支持多重继承，允许一个子类同时继承多个父类。

- 多重继承的定义格式为：
- class 子类名(父类名1,父类名2,...):
- 　类体
- 如果子类重新定义了构造方法，需要显式去调用父类的构造方法，此时调用哪个父类的构造方法由程序决定。

例：

```
class A:
    def __init__(self):
        print("AA")
class B:
    def __init__(self):
        print("BB")
class C(A,B):
    def __init__(self):
        B.__init__(self)
        A.__init__(self)
        print("CC")
def main():
    c=C()
main()
```

- 若子类没有重新定义构造方法，则只会执行第一个父类的构造方法。
- 若父类1，父类2，……中有同名的方法，通过子类的实例化对象去调用该方法时调用的是第一个父类中的方法。
- 对于普通的方法，其搜索规则和构造方法是一样的。

例：

```
class A:
    def __init__(self):
        print("AA")
    def __del__(self):
        print("aa")
class B:
    def __init__(self):
        print("BB")
    def __del__(self):
        print("bb")
class C(A,B):
    pass
def main():
    c=C()
main()
```

例：

```
class A:
    def __init__(self):
        print("AA")
    def __del__(self):
        print("aa")
class B:
    def __init__(self):
        print("BB")
    def __del__(self):
        print("bb")
class C(B,A):
    pass
def main():
    c=C()
main()
```

- 例：
- class A:
- 　　def f(self):
- 　　　　print("AA")
- class B:
- 　　def f(self):
- 　　　　print("BB")
- class C(A,B):
- 　　pass
- def main():
- 　　c=C()
- 　　c.f()
- main()

- 例：多重继承程序示例。
- class A():
- 　　def foo1(self):
- 　　　　print("AAAAA")
- class B(A):
- 　　def foo2(self):
- 　　　　print("BBBBB")
- class C(A):
- 　　def foo1(self):
- 　　　　print("CCCCC")
- class D(B,C):
- 　　pass
- d=D()
- d.foo1()

9.4.3 多态

- 多态即多种形态，是指不同的对象收到同一种消息时会产生不同的行为。
- 在程序中，消息就是调用函数，不同的行为就是指不同的实现方法，即执行不同的函数。

- Python中的多态和C++、Java中的多态不同，Python中的变量是弱类型的，在定义时不用指明其类型，它会根据需要在运行时确定变量的类型。
- 在运行时确定其状态，在编译阶段无法确定其类型，这就是多态的一种体现。

- Python本身是一种解释型语言，不进行编译，因此它就只在运行时确定其状态，故也可以说Python是一种多态语言。

- 在Python中，很多地方都可以体现多态的特性。
- 例：内置函数len()。
- len()函数不仅可以计算字符串的长度，还可以计算列表、元组等对象中的数据个数，此处，在运行时通过参数类型确定其具体的计算过程，正是多态的一种体现。

- 例：多态程序示例。
- class base(object):
- 　　def __init__(self,name):
- 　　　　self.name=name
- 　　def show(self):
- 　　　　print("base class:",self.name)
- class subclass1(base):
- 　　def show(self):
- 　　　　print("sub class 1:",self.name)

```
class subclass2(base):
    def show(self):
        print("sub class 2:",self.name)
class subclass3(base):
    pass
def testFunc(o):
    o.show()
first=subclass1("1")
second=subclass2("2")
third=subclass3("3")
lst=[first,second,third]
for p in lst:
    testFunc(p)
```

- 父类和多个子类中都有同名的方法，虽然方法同名，但该方法在不同类中的行为是不同的。
- 当向一个对象发送消息（即调用该方法）时，所得结果取决于是哪一个对象接收。
- 多个不同的对象都支持相同的消息，但各对象响应消息的行为不同，这种能力就是多态性的体现，即同一操作在不同的上下文环境中具有不同形态的意思。

例：

```
class A:
    def f(self):
        print("AA")
class B(A):
    def f(self):
        print("BB")
class C(B):
    def f(self):
        print("CC")
def main():
    A().f()
    B().f()
    C().f()
main()
```

例：

```
class A:
    def f(self):
        print("AA")
class B(A):
    def f(self):
        print("BB")
class C(A):
    def f(self):
        print("CC")
def main():
    A().f()
    B().f()
    C().f()
main()
```

9.5 面向对象程序设计应用举例

- 例：已知$y = \frac{f(40)}{f(30)f(20)}$，当

  f(n)=1×2+2×3+3×4+……+n×(n+1)时，求y的值。

- 面向过程方法的程序：

```
def f(n):       #定义求f(n)的函数
    s=0
    for i in range(1,n+1):
        s+=i*(i+1)
    return s
y=f(40)/(f(30)+f(20))
print('y=',y)
```

- 面向对象方法的程序：
- class calculate:
- def __init__(self,n):
- self.n=n
- def f(self):　　#求f(n)的成员函数
- s=0
- for i in range(1,self.n+1):
- s+=i*(i+1)
- return s

- ob1=calculate(40)
- ob2=calculate(30)
- ob3=calculate(20)
- y=ob1.f()/(ob2.f()+ob3.f())
- print('y=',y)

- 例：已知$y = 1 + \frac{1}{3} + \frac{1}{5} + \cdots + \frac{1}{2n-1}$，求：
- （1）y<3时的最大n值。
- （2）与（1）的n值对应的y值。
- 面向过程的程序：

- n=1
- y=0
- while y<3:
- 　f=1.0/(2*n-1)　　#求累加项
- 　y+=f　　#累加
- 　n+=1
- print("y={0},n={1}".format(y-f,n-2)) #退出循环时的y值和n值与待求y和n不同

- 面向对象的程序：
- class compute:
- def yn(self): #计算y和n的函数
- self.n=1
- self.y=0.0
- while self.y<3.0:
- self.f=1.0/(2*self.n-1)
- self.y+=self.f
- self.n+=1
- self.y=self.y-self.f
- self.n=self.n-2
- def print(self): #输出结果的函数
- print("y={0},n={1}".format(self.y,self.n))

```
def main():    #主函数
    obj=compute()
    obj.yn()
    obj.print()
main()
```

- 例：某商店销售某一商品，允许销售人员在一定范围内灵活掌握售价(price)，现已知当天3名销货员的销售情况为：

售货员号（num）	售货件数（quantity）	售货单价（price）
101	5	23.5
102	12	24.56
103	100	21.5

- 编写程序，计算当日此商品的总销售款sum以及每件商品的平均售价。
- 分析：利用字典来组织数据，以销货员号作为字典关键字，通过关键字遍历字典。

```
class Product:
    def total(self):
        prod={'101':[5,23.5],'102':[12,24.56],'103':[100,21.5]}
        self.sum=0.0
        self.n=0
        for key in prod.keys():
            quantity=prod[key][0]
            price=prod[key][1]
            self.sum+=quantity*price
            self.n+=quantity
    def display(self):
        print(self.sum)
        print(self.sum/self.n)
```

```
def main():
    ob=Product()
    ob.total()
    ob.display()
main()
```

自测题

- 一、选择题
- 1．下列说法中不正确的是（　　）。D
- A．类是对象的模板，而对象是类的实例
- B．实例属性名如果以__开头，就变成了一个私有变量
- C．只有在类的内部才可以访问类的私有变量，外部不能访问
- D．在Python中，一个子类只能有一个父类

- 2．下列选项中不是面向对象程序设计基本特征的是（　　）。C
- A．继承　　B．多态
- C．可维护性　　D．封装
- 3．在方法定义中，访问实例属性x的格式是（　　）。B
- A．x　　B．self.x
- C．self[x]　　D．self.getx()

- 4．下列程序的执行结果是（　　）。D
- class Point:
- 　　x=10
- 　　y=10
- 　　def __init__(self,x,y):
- 　　　　self.x=x
- 　　　　self.y=y
- pt=Point(20,20)
- print(pt.x,pt.y)
- A．10 20　　B．20 10　　C．10 10　　D．20 20

- 5．下列程序的执行结果是（　　）。A
- class C():
- 　　f=10
- class C1(C):
- 　　pass
- print(C.f,C1.f)
- A．10 10　　B．10 pass
- C．pass 10　　D．运行出错

- 二、填空题
- 1．在Python中，定义类的关键字是（　）。class
- 2．类的定义如下：
- class person:
- 　　name='Liming'
- 　　score=90

- 该类的类名是（　），其中定义了（　）属性和（　）属性，它们都是（　）属性。如果在属性名前加两个下划线（__），则属性是（　）属性。将该类实例化创建对象p，使用的语句为（　），通过p来访问属性，格式为（　）、（　）。
- person，name，score，公有，私有，p=person()，p.name,p.score

- 3．Python类的构造方法是（　），它在（　）对象时被调用，可以用来进行一些属性（　）操作；类的析构方法是（　），它在（　）对象时调用，可以进行一些释放资源的操作。__init__，生成，初始化，__del__，释放

- 4．可以从现有的类来定义新的类，这称为类的（　），新的类称为（　），而原来的类称为（　）、父类或超类。继承，子类，基类
- 5．创建对象后，可以使用（　）运算符来调用其成员。.

- 6．下列程序的运行结果为（　）。100
- class Account:
- 　　def __init__(self,id):
- 　　　　self.id=id
- 　　　　id=888
- acc=Account(100)
- print(acc.id)

- 7．下列程序的运行结果为（　）。100 100
- class parent:
- 　def __init__(self,param):
- 　　self.v1=param
- class child(parent):
- 　def __init__(self,param):
- 　　parent.__init__(self,param)
- 　　self.v2=param
- obj=child(100)
- print(obj.v1,obj.v2)

- 8．下列程序的运行结果为（　）。400
- class account:
- 　def __init__(self,id,balance):
- 　　self.id=id
- 　　self.balance=balance
- 　def deposit(self,amount):
- 　　self.balance+=amount
- 　def withdraw(self,amount):
- 　　self.balance-=amount
- acc1=account('1234',100)
- acc1.deposit(500)
- acc1.withdraw(200)
- print(acc1.balance)

- 三、简答题
- 1. 什么叫类？什么叫对象？它们有何关系？
- 2. 在Python中如何定义类与对象？
- 3. 类的属性有哪几种？如何访问它们？
- 4. 继承与派生有何关系？如何实现类的继承？
- 5. 什么是多态？在Python中如何体现？