**函数**

def函数名（形式参数）：

函数体(可以pass暂时不管)

**形参设置**

**默认值**

def 函数名（非默认参数，形参名=默认值，••••••）：

函数体

**可变长度参数**

在Python中, 函数可以接收不定个数的参数，即用户可以给函数提供可变长度的参数。这可以通过在形式参数前面使用标识符“ \* ”来实现。(这些参数被组合到一个元组上)(如果函数中形参是n个单变量，则在实参的序列变量名前加要求实参序列中 的元素个数与单变量形参个数相同；如果实参中普通变量与序列变量混用，则以“\* "为前 缀的序列变量放置在实参的最后。)

>>def all\_3(brgs, \* args):

print(brgs)

print(args)

>>all\_3('abc','a','b')

abc

(a,b)

在Python的函数形参中还提供了一种参数名前面加标识符“ ”的方式，用来 引用一个字典。函数调用者须以关键参数的形式为其赋值，形式参数得到一个以关键参数 中变量名为key,右边表达式值为value的字典。

>>def all\_4( \*\* args):

print(args)

>>all\_4(x='a',y='b',z = 2)

('y': 'b', 'x': 'a', 'z': 2)

**变量作用域**

def fun():

x=5牯函数内定义的局部变量，只能在函数内使用

global y牯函数内声明一个全局变量

**生成器函数**

Python中使用了 yield的函数返回生成器对象，此函数称为生成器函数，只能用于迭代 操作。在调用该函数的过程中，每次遇到yield语句时，函数会暂停执行，并保存当前所有的 运行状态信息，返回yield后面的值，并在下一次执行next()方法时从当前位置继续运行。 生成器函数返回的是一个生成器对象。

>>def fib(n):

i, a, b = 0, 0, 1

while i < n:

yield a#生成器返回对象是a

a, b = b, a + b

i += 1

>>n = 3

L=fib(n)#L是个生成器对象

>>for x in L:

print(x)

0

1

1

**lambda函数**

lambda函数是一个匿名函数，有时也被称为lambda表达式，比def格式的函数定义简 单很多。lambda函数可以接收任意多个参数，但只返回一个表达式的值。lambda中不能 包含多个表込式。

#lambda函数的定义格式为：(其中形式参数可以有多个，它们之间用逗号隔开。表达式只有一个。返回表达式的计算 结果。)

lambda 形式参数：表达式

**类与对象**

类和对象的关系:

(1) 类是对象的抽象，而对象是类的具体实例。

(2) 类是抽象的，而对象是具体的。

(3) 每一个对象都是某一个类的实例。

(4) 每一个类在某一时刻都有零或更多的实例。

(5) 类是静态的，它们的存在、语义和关系在程序执行前就已经定义好了，对象是动态 的，它们在程序执行时可以被创建和删除。

(6) 类是生成对象的模板。(存在get和set)

**类的格式**

一个类中有两个特殊的方法：\_\_new\_\_()和―init\_\_()0这两个方法合起来类似于Java 和C++等程序设计语言中的构造方法，用于创建并初始化一个对象。当实例化(创建)一个 类对象时，最先被调用的是\_\_new\_\_()方法。\_\_new\_\_()方法用于创建对象。\_\_new\_\_()方 法创建完对象后，将该对象传递给\_\_init\_\_()方法中的self参数。而\_\_init\_\_()方法是在对 象创建完成之后初始化对象状态。这两个方法都是在实例化对象时被自动调用的，不需要 程序显式调用。关于\_\_new\_\_()方法将在8.4节中进一步介绍。

如果用户未设计\_init\_\_()方法,Python将提供默认的 \_init\_()方法。每个方法 其实都是一个函数定义，与普通函数略有差别。

(1) 每个实例方法的第一个参数都是self,self代表将来要创建的对象实例本身。在访 问类的实例属性时需要以self为前缀。

(2) 实例方法只能通过对象来调用，即向对象发消息请求对象执行某个方法。

class 〈类名〉：

类属性1

...

类属性n

〈方法定义1>

...

〈方法定义n>

注：“单下画 线”开始的属性叫作保护属性，只有其本身和子类能访问到这些属性。“双下画线”开始的是 私有属性，只有该类对象能访问，即使子类对象也不能访问到这个属性。

class Person (object):#object表示的是父类名(默认是object)

number\_s = 0

\_\_number=0

def \_\_new\_\_ (cls,n, y, w, h):#\_\_new\_\_方法一般不重写

print('执行\_\_new\_\_方法')

return super (Person, cls). \_\_new\_\_(cls)

def \_\_init\_\_(self, n, y, w, h):

self.name = n

self.year = y

self.\_\_weight = w#甘定义私有属性以千克为单位的体重weight

self.\_\_height = h#甘定义私有属性以米为单位的身高height

def old(self, y):

return y - self.year

@staticmethod#声明静态，去掉则编译报错

def getNumber\_s() :# 静态方法没有 self

print("总人数为："，Person, number\_s)

#静态方法只能访问属于类的成员，不能访问 属于对象的成员。

#使用静态方法时，既可以通过“对 象名.静态方法名”来访问，也可以通过“类名.静态方法名”来访问。

@classmethod

def getNumber\_c(cls):#类方法定义之前由“ @classmethod”语句引导，第一个形参通常被命名为cls。类方法 既可以通过类名，也可以通过对象名来调用。

return cls.\_\_number

def \_\_del\_\_(self):#Python中类的析构方法是\_\_del\_\_()方法，用来删除对象以释放对象的内存空间。如 果用户未提供析构方法，Python将提供一个默认的析构方法。

print('调用删除对象方法')

**类的重用**

类的继承是指在现有类的基 础上创建新类，在新类中添加代码，以扩展原有类的属性（数据成员）和方法（成员函数）。类 的组合是指在新创建的类中包含有已有类的对象作为其属性。

**类的继承**

继承的出发点在于一些类存在相似点，这些相似点可以被提取出来构成一个基类，基类 中的代码通过继承可以在其他类中重用。继承是在一个被作为父类（或称为基类）的基础上 扩展新的属性和方法来实现的。父类定义了公共的属性和方法，继承父类的子类自动具备 父类中的非私有属性和非私有方法，不需要重新定义父类中的**非私**有内容，并且可以增加新 的属性和方法。

class ChildClassName(ParentClassNamel [, ParentClassName2[, ParentClassName3, •••]])：

于类体或pass语句/

**类的组合**

类的组合(composition)是类的另一种重用方式。如果程序中的类需要使用其他类的 对象,就可以使用类的组合方式。组合关系可以用“has-a”关系来表达，就是一个主类中包 含其他对象。

在继承关系中，父类的内部细节对于子类来说在一定程度上是可见的。所以通过继承 的代码复用可以说是一种“冃盒式代码复用”。在组合关系中，对象之间各自的内部细节是 不可见的，所以通过组合的代码复用可以说是一•种“黑盒式代码复用”。

class Display (object):

pass

class Memory (object):

pass

class Disk (object):

pass

class Processor(object):

pass

class Computer(object):

def \_\_init\_\_(self):

self, display = Display()

self, memory = Memory()

self.disk = Disk()

self.processor = Processor()