**Python面向对象：**

**类：**

**定义**：类是抽象的模板

实例是根据类创建出来的一个个具体的“对象”，每个对象都拥有相同的方法，但各自的数据可能不同

**编写**：类通过**class**关键字定义，依次写出类名、括号、括号中的继承的类名和冒号:，

如：

class Person(object):

Pass

创建实例：使用 类名+（） 如：xiaomi = Person()

**属性：**

属性也分实例属性和类属性

实例属性每个实例各自拥有，互相独立，而类属性有且只有一份;

当实例属性和类属性重名时，实例属性优先级高，它将屏蔽掉对类属性的访问。

定义类属性：

//直接在 class 中定义：

如：class Person(object):

address = 'Earth'

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

Person.address = ‘aaa’

self.name = name

设置属性： 除了可以直接使用self.name = 'xxx'设置一个属性外

还可以通过 setattr(self, 'name', 'xxx') 设置属性

访问：类的方法和属性，使用点（.）

属性权限的控制：通过属性名来实现的，如果一个属性由双下划线开头(\_\_)，该属性就无法被外部访问

（但是，如果一个属性以"\_\_xxx\_\_"的形式定义，那它又可以被外部访问了，以"\_\_xxx\_\_"定义的属性在Python的类中被称为特殊属性）

**方法：**

方法也分实例方法和类方法

定义类方法：通过标记一个**@classmethod**，该方法将绑定到类上。类方法的第一个参数将传入类本身，通常将参数名命名为 cls，上面的 cls.count 实际上相当于 Person.count。

如：

class Person(object):

count = 0

@classmethod

def how\_many(cls):

return cls.count

定义实例方法：在class中定义的全部是实例方法，实例方法第一个参数 self 是实例本身。

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

self.name = name

Person.count = Person.count + 1

\_\_init\_\_() 方法：当创建实例时，\_\_init\_\_()方法被自动调用，的第一个参数必须是 self，后续参数则可以自由指定，和定义函数没有任何区别。

**类的继承：**

**定义**：当我们定义一个class的时候，可以从某个现有的class继承，新的class称为子类，而被继承的class称为父类或超类；子类获得了父类的全部功能

父类：

class Person(object):

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

self.name = name

子类：

class Student(Person):

def \_\_init\_\_(self, name, score):

super(Student, self).\_\_init\_\_(name)

self.score = score

一定要用**super(Student, self).\_\_init\_\_(name)**去初始化父类，函数super(Student, self)将返回当前类继承的父类，即 Person ，然后调用\_\_init\_\_()方法

**获取对象信息：**

 isinstance（要判断的实例，类型） ：判断它是否是某种类型的实例，返回bool类型

type（变量）： 函数获取变量的类型

 dir（变量）： 函数获取变量的所有属性，包括`\_\_class\_\_`这类有特殊意义的属性

**类的多态：**

**定义：**子类的run()覆盖了父类的run()，在代码运行的时候，总是会调用子类的run()

（总是先查找它自身的定义，如果没有定义，则顺着继承链向上查找，直到在某个父类中找到为止）

（当从多个超类继承相同名字的不同方法是，圆括号内的超类名字位于前面的会覆盖位于后面的方法）

动态语言和静态语言（例如Java）最大的差别之一。动态语言调用实例方法，不检查类型，只要方法存在，参数正确，就可以调用。

**多重继承**：除了从一个父类继承外，Python允许从多个父类继承，称为多重继承。

Python的特殊方法：

特殊方法定义在class中，不需要直接调用，Python的某些函数或操作符会自动调用对应的特殊方法