概念

OOP把对象作为程序的基本单元，一个对象包含了数据和操作数据的方法函数。（更复杂的Object）(改变对象的状态来达到目的，与函数式编程截然相反)**多个对象之间的协同交互**

自定义的**对象数据类型**就是面向对象中的类（Class）的概念。（泛化->例化）总结对象的通用行为，即共性，在例化时再赋予其个性。

如果采用面向对象的程序设计思想，我们首选思考的不是程序的执行流程，而是Student这种数据类型应该被视为一个对象，这个对象拥有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print\_score消息，让对象自己**做出相应行为**，把自己的数据打印出来。

仍以Student类为例，在Python中，定义类是通过class关键字：

**class Student(object):**

**pass**

class后面紧接着是类名，即Student，类名通常是大写开头的单词，紧接着是(object)，表示该类是从哪个类继承下来的

定义好了Student类，就可以根据Student类创建出Student的实例，创建实例是通过类名+()实现的

注意到\_\_init\_\_方法的第一个参数永远是self，表示创建的实例本身，因此，在\_\_init\_\_方法内部，就可以把各种属性绑定到self，因为self就指向创建的实例本身。

有了\_\_init\_\_方法，在创建实例的时候，就不能传入空的参数了，必须传入与\_\_init\_\_方法匹配的参数，但self不需要传，Python解释器自己会把实例变量self传进去

{

**注意method其实有两种，一种是instance method，一种是class method：**

instance method，就相当于对于机器人这个产品进行各种修改的车间。我给机器人上色，不影响我这个工厂的外形对吧？

class method，是对这个工厂，这个class的属性进行修改的车间，比如我有一个车间负责把工厂涂成红色的，这个行为并不影响我造出来的机器人的大小颜色属性。**如果在method里面要改变instance的属性，可以用self.xxxx来指代这个属性进行修改**。

同理，**变量也分为实例变量（self.name）与类变量**。

}

class Student:

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

self.name=name

self.number=number

如果没有初始化，则例化时不需要传参

Self是什么：

具体来说，是该实例的内存地址。

类似的，如果实例化ins后，执行ins.fun1( )，python解释器会将ins.fun1( )解释成Test.fun1(ins)

**Self体现了类中的例化部分**

三大思想

封装：

私有化 双下划线：\_\_name 如果想外部代码要获取name，应使用专用方法得到

继承与多态：

继承的主要用途是实现多态（why？），重要的是接口继承性（流行基于接口的轻度继承，目的是理清关系）

多态即就算不知道变量或参数的数据类型也能进行操作，比如+，对于整型与字符串都是有定义的

你会发现，新增一个Animal的子类，不必对run\_twice()做任何修改，实际上，任何依赖Animal作为参数的函数或者方法都可以不加修改地正常运行，原因就在于多态，我们只需要接收Animal类型就可以了，然后按照Animal类型进行操作即可（因为run在animal类型下必有定义，而各个子类可以根据情况自定义具体实现方式，这便是**基于接口的轻度继承**）。

对于一个变量，我们只需要知道它是Animal类型，无需确切地知道它的子类型，就可以放心地调用run()方法，而具体调用的run()方法是作用在Animal、Dog、Cat还是Tortoise对象上，由运行时该对象的确切类型决定，这就是多态真正的威力：调用方只管调用，不管细节，而当我们新增一种Animal的子类时，只要确保run()方法编写正确，不用管原来的代码是如何调用的。

**这就是著名的“开闭”原则：**

**对扩展开放：允许新增Animal子类**

**对修改封闭：不需要修改依赖Animal类型的run\_twice()等函数。**

对于Python这样的动态语言来说，则不一定需要传入Animal类型。我们只需要保证传入的对象有一个run()方法就可以，这就是动态语言的“鸭子类型”，它并不要求严格的继承体系，一个对象只要“看起来像鸭子，走起路来像鸭子”，那它就可以被看做是鸭子（动态语言不提供静态语言提供的类型安全特性），为了自动检查，可以加入类型提示

def run\_twice(animal: Animal):。

多态简单来说，可以理解为一种接口，多种实现，从而提高了代码的扩展性，接口的复用性 ，而其有几个必要条件：继承，重写，子类引用父类

Ex. Count increments when a new instance is created

class Student(object):

count=0

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

self.name=name

Student.count=Student.count+1

Ps: count+=1不可，

如果不放init()中，每次都会初始化count，count得不到修改。且因为\_\_init\_\_方法里已经修改了类属性，故每次不会重新赋初值