



2 years ago



## 1/8

从机器的角度理解世界，还是纯数学的方式理解世界，都不能代表人的视角和思维方式。在二者相向发展中，有了很多交汇。计算机当然离理解人的思维方式还差的很远，除非把人变成计算单元。科学家们在这个方向上努力良久，在编程语言领域“面向对象”的思想和方法被广泛接受。事物是不断变化的，人类在变化中寻找相对静止的时空来思考世界，来描述世界，文字、绘画都是语言，都需要在静止中呈现。生命尊重并表现自我，认同个体，于是世间有了物的概念。在静止中，如果还只是思考数据，那么就是混沌，观察个体才有意义，才有血肉。世间万物，物就是对象。

所谓面向对象，就是把你眼中能认为或抽象出的独立事务描述清楚，首先它是个整体，然后我们再肢解它，最后在把它重新放到变化中观察行为。抽象的越彻底，我们越能发现很多事务的共性，于是有了分类。在变化中，物与物必然会产出影响，于是有了关系。

视角发生了变化，描述事物和行为的方式必然有了变化，产生了新的表达方法，新的技巧，同时也有了新的问题和挑战，当然会产生新的解决问题的方法，这些就是面向对象的基本方法，设计模式，架构经验，等等。

下面我们进入Python的面向对象世界。

新建oo.py文件，用于测试和练习。

## 1.8.1 类与对象

在python中，我们使用class关键字来定义一类事物，类是一个抽象描述，并不是真正的存在，需要把它初始化才会产生一个真正的事物，我们称之为对象。在编程过程中，拥有行为和数据的对象，而不是类。

下面我们声明一个简单的类：

```
# -*- coding: UTF-8 -*-  
  
class Person:  
    pass # An empty block  
  
p = Person()  
print p
```

我们使用class语句后跟类名，创建了一个新的类。这后面跟着一个缩进的语句块形成类体。在这个例子中，我们使用了一个空白块，它由pass语句表示。接下来，我们使用类名后跟一对圆括号来创建一个对象/实例。为了验证，我们简单地打印了这个变量的类型。它告诉我们我们已经在\_\_main\_\_模块中有了一个人Person类的实例。结果如下：

```
<__main__.Person object at 0x10a13f630>
```

你可能已经注意到存储对象的计算机内存地址也打印了出来。这个地址在你的计算机上会是另外一个值，因为Python可以在任何空位存储对象。

## 1.8.2 对象的方法

方法代表现实世界中的行为，简单示例如下：

```
class Person1:
    def sayHi(self):
        print('Hello, how are you?')

p1 = Person1()
p1.sayHi()
```

这里我们需要注意sayHi方法没有任何参数，但仍然在函数定义时有self，self代表对象本身，等价于C++中的self指针和Java、C#中的this引用。之后，我们通过对象名加点的方式来调用对象的方法。运行结果如下：

```
Hello, how are you?
```

## 1.8.3 构造函数

我们在使用类来构造一个对象的时候，通常在构造之初需要把对象本身的关键属性进行初始化。比如初始化一个人，如果我们认为年龄和名字是关键属性的话，我们在实例化这个人的时候，需要把相关属性的值传进来。上面我们知道初始化对象的时候是类名加括号的方式，实际上这个括号调用了一个内置的方法，我们称之为构造函数，正是该函数返回了被初始化的对象本身。

python的构造函数名为\_\_init\_\_，我们可以自定义传入参数的类型和个数。示例如下：

```
class Person2:
    def __init__(self, name):
        self.name = name
    def sayHi(self):
        print('Hello, my name is', self.name)

p2 = Person2('玄魂')
p2.sayHi()
```

这里，我们把\_\_init\_\_方法定义为取一个参数name（以及普通的参数self）。在这个\_\_init\_\_里，我们只是创建一个新的字段 name。最重要的是，我们没有专门调用\_\_init\_\_方法，只是在创建一个类的新实例的时候，把参数包括在圆括号内跟在类名后面，从而传递给\_\_init\_\_方法。

现在，我们能够在我们的方法中使用self.name变量，这在sayHi方法中得到了验证，运行结果如下：

```
Hello, my name is 玄魂
```

## 1.8.4 变量

变量代表属性和数据。在python中变量有两种，类变量和对象变量。类变量是全局的，对每个对象都共享，对象的变量是实例化的，每个对象之间互不影响。下面我们通过一段代码来对比：

```
class Person3:
    '''Represents a person.'''
    population = 0

    def __init__(self, name):
        '''Initializes the person's data.'''
        self.name = name
        print( '(Initializing %s)' % self.name)

        # When this person is created, he/she
        # adds to the population
        Person3.population += 1

    def sayHi(self):
        '''Greeting by the person.

        Really, that's all it does.'''
        print('Hi, my name is %s.' % self.name)

    def howMany(self):
        '''Prints the current population.'''
        if Person3.population == 1:
            print('I am the only person here.')
        else:
            print('We have %d persons here.' % Person3.population)

xh = Person3('玄魂')
xh.sayHi()
xh.howMany()

kl = Person3('🐼考拉')
kl.sayHi()
kl.howMany()

xh.sayHi()
xh.howMany()
```

这个例子略微有点长，但是它有助于说明类与对象的变量的本质。这里，population属于Person类，因此是一个类的变量。name变量属于对象（它使用self赋值）因此是对象的变量。

观察可以发现\_\_init\_\_方法用一个名字来初始化Person实例。在这个方法中，我们让population增加1，这是因为我们增加了一个人。同样可以发现，self.name的值根据每个对象指定，这表明了它作为对象的变量的本质。

在这个程序中，我们还看到docstring对于类和方法同样有用。我们可以在运行时使用Person.\_\_doc\_\_和Person.sayHi.\_\_doc\_\_来分别访问类与方法的文档字符串。

这里需要注意的是：

1. Python中所有的类成员（包括数据成员）默认都是公有的。
2. 如果你使用的数据成员名称以双下划线前缀比如\_\_privatevar，Python的名称管理体系会有效地把它作为私有变量。
3. 这样就有一个惯例，如果某个变量只想在类或对象中使用，就应该以单下划线前缀。而其他的名称都将作为公共的，可以被其他类/对象使用。记住这只是一个惯例，并不是Python所要求的（与双下划线前缀不同）。

## 1.8.5 继承

继承这个概念，经常以父与子的关系来被阐述，我认为这阐述对初学者来说是一种误导。在人类社会中，父与子继承的是什么？是财产，实际是赠予。面向对象的继承，实际是不是从现实世界中来，而是为了解决面向对象编程的问题而产生的，要解决的是代码复用的问题。比如人类有许多共性，有鼻子，有眼睛。但是人类也是分亚种的，从肤色上划分人种，地理上划分人群。那么在定义黄种人和黑种人的时候，不可避免的要重新定义很多属性和行为，于是有了继承的概念，把公共的内容放到Person类里面，然后AsiaPerson继承Person类，它自然就有了Person类定义的内容，这种思想，其实换个词可能更好理解——克隆。概念就不纠结了，理解就好，下面看代码：

```
class SchoolMember:
    '''Represents any school member.'''
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print('(Initialized SchoolMember: %s)' % self.name)

    def tell(self):
        '''Tell my details.'''
        print('Name:"%s" Age:"%s"' % (self.name, self.age))

class Teacher(SchoolMember):
    '''Represents a teacher.'''
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print('(Initialized Teacher: %s)' % self.name)

    def tell(self):
        SchoolMember.tell(self)
        print('Salary: "%d"' % self.salary)
```

```
class Student(SchoolMember):
    '''Represents a student.'''
    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
        self.marks = marks
        print('(Initialized Student: %s)' % self.name)

    def tell(self):
        SchoolMember.tell(self)
        print('Marks: "%d"' % self.marks)

t = Teacher('Mrs. Shrividya', 40, 30000)
s = Student('xh', 22, 75)

print() # prints a blank line

members = [t, s]
for member in members:
    member.tell() # works for both Teachers and Students
```

为了使用继承，我们把基类的名称作为一个元组跟在定义类时的类名称之后。然后，我们注意到基类的\_\_init\_\_方法专门使用self变量调用，这样我们就可以初始化对象的基类部分。这一点十分重要——Python不会自动调用基类的构造函数。调用基类的构造函数需要使用基类名进行调用，而且传入子类的self。

注意，在我们使用SchoolMember类的tell方法的时候，我们把Teacher和Student的实例仅仅作为SchoolMember的实例。

另外，在这个例子中，我们调用了子类型的tell方法，而不是SchoolMember类的tell方法。可以这样来理解，Python总是首先查找对应类型的方法，在这个例子中就是如此。如果它不能在导出类中找到对应的方法，它才开始到基本类中逐个查找。

python支持多继承，可以同时继承多个基类。

## 1.8.6 小结

Python是一个高度面向对象的语言，我们只是过了一些基本概念，万物皆对象的理念，也让高级的python编程变得更加有趣。

本节练习题如下：

1. 创建一个动物基类，子类老虎，🐱公鸡继承该类。实现基本属性的初始化，🐱包括种类，🐱颜色等。实现基本的🐱奔跑、吃饭等方法。

下一节开始我们正式进入Python系统级编程实践。

本系列教程全部内容在星球空间内发布，并提供答疑和辅导。





星主：程序员－玄魂

星球：玄魂工作室-安全圈



知识星球

长按扫码预览社群内容  
和星主关系更近一步

欢迎到关注微信订阅号，交流学习中的问题和心得

