Table of Contents ¶

1 面向对象编程

In [2]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

面向对象和面向过程编程

面向过程编程和面向对象编程是编程过程中不同的思维方式

1.1 面向过程

1.1.1 什么是过程?

过程是指事物发展所经过的程序,即事物发展的先后经过,先做什么,再做什么,接着做什么等等

生活中的过程:比如吃饭,思考一个完整的吃饭过程应该是怎样的?

吃饭最开始的流程应该从播种开始,网上曾经有个博主再现了一次从播种到收割然后自己制作锅制作刀最 后做完一道菜的完整流程,这就是过程

1.1.2 什么是面向过程?

面向过程是一种解决问题的思路

使用这种思路,需要关注整个过程中的每一步,使用这种思路编程,需要关注最终问题的解决需要哪几步,然后针对每一步编写程序实现对应的功能,最后根据功能的先后顺序一次调用,实现解决最终问题的目的.

基于该思想编写程序就好比在编写一条流水线，是一种机械式的思维方式:

- 优点：复杂的问题流程化、进而简单化，从而按照流程解决问题即可 (需求也简单的情况下,优点会更明显)

- 缺点：如果需求复杂功能太多,面向过程编程会变的非常复杂.比如一个问题需要几百个步骤解决,中间任何一个步骤出现任何问题都会影响整个功能的实现,开发时困难,维护时更困难
- 总结：面向过程编程的思想就是，对于一个问题，你的内心必须知道解决问题的所有步骤，之后严格按照该步骤，从而达到解决问题的目的。

1.2 面向对象

1.2.1 什么是对象?

对象是实际存在的个体,python中一切皆对象 生活中的对象:一张桌子,一支笔,一个人等等,万物皆对象

1.2.2 什么是面向对象?

面向对象也是一种解决问题的思路

使用这种思路解决问题,关注点在问题的参与者,即整个需求中有哪些问题需要解决,这些问题有没有已经 写好的程序可以直接解决,而不再是具体到每一步需要写怎样的程序来解决.所以在面向对象编程中,更多的是在调用别人写好的功能

面向过程和面向对象的区别

区别	面向过程	面向对象
关注点	每一步功能怎么实现	每一步功能是否已经有人写好可以直接拿来用
效率	参与每一步的实现,效率比较慢	大多数调用别人写好的功能,效率较快
质量	每一步亲力亲为,出错几率较高	大多数调用的功能都经过大量验证,不易出错
适用范围	适合解决需求简单的问题	适合解决复杂需求

面向对象是基于面向过程的

1.2.3 面向对象的三大特征

- 封装 封装一种将抽象性函数接口的实现细节部分包装、隐藏起来的方法.适当的封装可以让代码更容易理解与维护，也加强了代码的安全性。
- 继承 是子类继承父类的特征和行为，使得子类对象（实例）具有父类的实例域和方法，或子类从父类继承方法，使得子类具有父类相同的行为。
- 多态 多态是指同一个方法调用由于对象不同会产生不同的行为

2 类和对象

2.1 类

2.2 什么是类?

类是抽象的模板,用来描述具有相同属性和方法的对象的集合

所谓属性,通俗的可以理解为特征,而方法是可操作性的技能,例如列表中进行删除和增加元素的方法.

2.3 先有类还是先有对象?

- 在现实世界中：一定是先有对象，后来随着智慧生物的发展归纳总结出来的类。对象是实际存在的，而类只是一种抽象概念
- 在程序中,务必保证：先定义类，后面再通过类实例化对象 (从一个类里面具体拿出一个对象的过程叫做实例化)

2.4 定义一个类

类由三部分组成:

- 类的名称:类名 ,通常采用驼峰式命名,即**首字母大写**,尽量让字面意思体现出类的特点
- 类的属性:一组数据(即该类有哪些特征)
- 类的方法:允许对该类进行操作的方法或行为 在定义一个类的时候,类名是必选项,属性和方法是可选项,在定义类的过程中,也可以加写类的帮助文档

```
class 类名:
    "类的帮助信息"
    "类的属性"
    "类的方法"
```

In [4]:
```
class Cook:  #类名后面不需要加括号
    '这是一个厨师的类'  #类的帮助文档
    #用变量表示属性
    name='张三'
    age='25岁'

    #用函数表示方法
    def qie():
        print('技能:切菜')

    def chao():
        print('技能:炒菜')
```

这样就成功定义了一个最简单的类，但是其中有一个明显的缺点是该厨师类中，所有的厨师的名字和年龄都相同,这个问题暂留悬念后续解决.

2.5 查看类的说明文档

In [5]:
```
Cook()    #光标点在括号里,然后shift+tab直接查看
```

Out[5]: <__main__.Cook at 0x1c27c070280>

In [7]:
```
Cook.__doc__  #首尾双下划线表示定义特殊的方法,一般是系统定义的名字
```

Out[7]: '这是一个厨师的类'

2.6 类属性的操作

2.6.1 查看类属性

查看类属性的方法是:
类名.属性名   #属性名后面不需要加括号

In [8]:
```
Cook.name    #查看属性,不需要有括号
```

Out[8]: '张三'

In [9]:
```
Cook.age
```

Out[9]: '25岁'

2.6.2 增加类属性

In [10]:
```
Cook.height='178cm'
```

In [11]:
```
Cook.height
```

Out[11]: '178cm'

2.6.3 删除类属性

In [12]:
```
del Cook.height
```

In [13]:
```
Cook.height  #这个属性被删除了,所以不存在了,就查不到了
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23336/3553436306.py in <module>
----> 1 Cook.height

AttributeError: type object 'Cook' has no attribute 'height'
```

2.6.4 修改类属性

In [14]:
```
Cook.age='28岁'
```

In [15]:
```
Cook.age
```

Out[15]: '28岁'

2.6.5 调用类的方法

```
调用类的方法是:
类名.方法名()      #方法名的后面括号要加上
```

In [17]:
```
Cook.qie()  #调用类的方法需要有括号
```

```
技能:切菜
```

In [18]:
```
Cook.chao()
```

```
技能:炒菜
```

2.7 类的实例化---对象

2.7.1 区分类和对象

以下哪些是类,哪些是对象?

- 飞机 波音737 空客公司编号89757的波音737
- 笔记本 华为笔记本 我正在用的这台华为笔记本
- 宝马 宝马X5 车牌号为京A12345的宝马X5

```
类:飞机,波音737
```

```
    笔记本,华为笔记本
    宝马,宝马X5
对象:空客公司编号89757的波音737
    我正在用的这台华为笔记本
    车牌号为京A12345的宝马X5
```

实例化

在面向对象的编程中，把用类创建对象的过程称为实例化,是将一个抽象的概念类，具体到该类实物的过程。

In [19]:
```
cook1=Cook()
```

In [20]:
```
cook1
#是从kCook类里实例化出来某一个对象存放在某一个地方,看不到它具体的内容
#只能通过对象调用对象的属性和方法
```

Out[20]:
```
<__main__.Cook at 0x1c27ddc1c40>
```

In [21]:
```
cook2=Cook()
```

In [22]:
```
id(cook1)
id(cook2)    #id不一样说明cook1和cook2是实例化出来的不同的对象
```

Out[22]:
```
1934846860352
```

Out[22]:
```
1934846860064
```

这里实例化出的对象,不一定是符合要求的,就像从人这个类中实例化一个具体的人出来,大多都想实例化 一个好看的,从其他类里实例化对象也会想要符合自己需求的对象,如何才能实例化一个符合要求的对象 出来呢?暂留悬念,后续解决

2.7.2 通过对象调用属性和方法

面向对象编程里面有一句非常经典的描述，通过类实例化一个对象，通过对象调类的方法。

In [23]:
```
cook1.name
cook2.name
```

Out[23]: '张三'

Out[23]: '张三'

In [24]:
```
cook1.age
cook2.age
```

Out[24]: '28岁'

Out[24]: '28岁'

In [26]:
```
cook1.age='30岁'    #可以更改已经实例化后对象的属性
```

In [27]:
```
cook1.age  #因为cook1和cook2的id不一样,是实例化出来的两个对象,所以修改cook1的对象,cook2不会被修改
cook2.age
```

Out[27]: '30岁'

Out[27]: '28岁'

In [28]:
```
cook1.qie()
#报错,通过实例化调用对象后,再去看它的方法,会有一个自动传参的过程
#所以在调用类的方法的时候,需要传入一个形参
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23336/4059402728.py in <module>
----> 1 cook1.qie()

TypeError: qie() takes 0 positional arguments but 1 was given
```

2.8 定义类中的方法时，必须传入一个形参

In [29]:
```
class Cook1:  #类名后面不需要加括号
    '这是一个厨师的类'  #类的帮助文档
    #用变量表示属性
    name='张三'
    age='25岁'

    #用函数表示方法
    def qie(self):    #所谓形参就是用一个函数名进行占位,这里用self进行一个占位
        print('技能:切菜')

    def chao(self):    #所谓形参就是用一个函数名进行占位,这里用self进行一个占位
        print('技能:炒菜')
```

In [30]:
```
cook3=Cook1()
cook4=Cook1()
```

In [31]:
```
cook3.name
cook3.age  #通过实例化对象后,调用对象的属性都是没有问题的
```

Out[31]: '张三'

Out[31]: '25岁'

In [32]:
```
cook4.name
cook4.age   #通过实例化对象后,调用对象的属性都是没有问题的
```

Out[32]: '张三'

Out[32]: '25岁'

In [33]:
```
cook3.qie()
cook4.qie()
```

```
技能:切菜
技能:切菜
```

In [34]:
```
cook3.chao()
cook4.chao()
```

```
技能:炒菜
技能:炒菜
```

解密时间:注意观察，类Cook1里面的两个函数qie()和chao():

In [35]:
```
class Cook2:  #类名后面不需要加括号
    '这是一个厨师的类'  #类的帮助文档
    #用变量表示属性
    name='张三'
    age='25岁'

    #用函数表示方法
    def qie(self):    #所谓形参就是用一个函数名进行占位,这里用self进行一个占位
        print('技能:切菜')
        print('参数self来自于:',id(self))

    def chao(self):   #所谓形参就是用一个函数名进行占位,这里用self进行一个占位
        print('技能:炒菜')
```

这两个函数的都有一个参数self,而且不是默认参数，并没有传入默认值，理论上在调用这个函数的时候 就必须传入一个实参，然而在通过对象调用方法时并没有传入任何实参，说明Python的内在机制里面会 有一个自动传参的过程。 比如下面代码，并没有为函数chao()传入任何实参，但是依然完美执行，说明在调用方法时python内部 把某个东西传给了形参self,那么传给了self的是什么呢?

在调用类的方法的时候，通过实例化出来的对象进行调用，其实self接收的实例化出来的对象本身！下 边进行验证:

In [36]:
```
cook5=Cook2()
cook6=Cook2()
```

In [37]:
```
cook5.qie()
```

```
技能:切菜
参数self来自于: 1934846858576
```

In [38]:
```
id(cook5)
```

Out[38]: 1934846858576

In [39]:
```
cook6.qie()
```

```
技能:切菜
参数self来自于: 1934846857712
```

In [40]:
```
id(cook6)
```

Out[40]: 1934846857712

通过实例化对象调用方法的时候,内部有一个自动化传参的过程,
会把该对象传给第一个参数的位置,所以我们在类中定义函数(方法)的时候,第一个参数必须写上
这个参数用来接收实例对象

重点，在类中定义函数(方法)的时候，第一个参数必须写上，这个参数用来接收实例对象。

2.9 类属性和实例属性

2.9.1 类属性

类属性是指定义在类中，但又在函数体外的属性。类属性可以被类的所有实例所共享。 对于下面这个类的两个属性name和age就是类属性，对于该类实例化的对象都是一样的,但是往往是我 们需要实例化出属性不同的对象。

解密第一个悬念

实例化类的时候需要一个初始化过程,用特殊的函数 **init**() 来初始化功能,这个特殊函数也可以称 作是构造函数。

每当创建一个新的实例的时候，python会自动执行这个函数。

和其他函数一样，这个构造函数也必须有一个参数接收实例对象本身，并且必须是第一个参数。 与类中其他函数不同的时,用来初始化的函数在实例化对象的时候自动运行,不需要使用对象进行调用

In [44]:
```
class Cook3:
    #用变量表示属性
    name='张三'
    age='25岁'

    #特殊函数(构造函数)
    def __init__(seft):    #类中定义的函数第一个参数用于接收实例对象,__init__函数也一样
        print('__init__函数在实例化对象的时候会被自动执行')

    #用函数表示方法
    def qie(seft):
        print('技能:切菜')

    def chao(seft):
        print('技能:炒菜')
```

In [45]:
```
cook7=Cook3()  #__init__()函数在实例化对象的时候会被自动执行,其他函数不会
```

```
__init__函数在实例化对象的时候会被自动执行
```

In [46]:
```
cook8=Cook3()  #每次实例化对象都会被自动执行
```

```
__init__函数在实例化对象的时候会被自动执行
```

2.9.2 实例属性

需要实例化不同属性的对象时，可以把这些需要改变的属性定义在init()函数内部，这种属性可以称之为 实例属性。 在实际实例化的时候需要把指定值传递给这些属性,就像函数一样

In [74]:
```
class Cook4:
    #用变量表示属性 类属性
    country='中国'

    #特殊函数(构造函数) 设置实例属性
    def __init__(self,name,age):    #类中定义的函数第一个参数用于接收实例对象,__init__函数也一样
        self.name=name
        self.age=age    #self.name和self.age都是给实力属性定义的名字
        print('__init__函数在实例化对象的时候会被自动执行')

    #用函数表示方法
    def qie(self):
        print('技能:切菜')

    def chao(self):
        print('技能:炒菜')
```

In [48]:
```
cook9=Cook4()   #需要写入两个参数 name和age
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23336/339330995.py in <module>
----> 1 cook9=Cook4()

TypeError: __init__() missing 2 required positional arguments: 'name' and 'age'
```

In [75]:
```
cook9=Cook4('李四','30岁')
```

```
__init__函数在实例化对象的时候会被自动执行
```

In [76]:
```
cook9.country
```

Out[76]:
```
'中国'
```

In [77]:
```
cook9.name
cook9.age
```

Out[77]: '李四'

Out[77]: '30岁'

In [78]:
```
cook10=Cook4('王五','26岁')
```

__init__函数在实例化对象的时候会被自动执行

In [79]:
```
cook10.country
```

Out[79]: '中国'

In [80]:
```
cook10.name
cook10.age
```

Out[80]: '王五'

Out[80]: '26岁'

In [81]:
```
class Cook5:
    #用变量表示属性 类属性
    country='中国'

    #特殊函数(构造函数) 设置实例属性
    def __init__(self,name,age):  #类中定义的函数第一个参数用于接收实例对象,__init__函数也一样
        self.name1=name
        self.age1=age
        print('__init__函数在实例化对象的时候会被自动执行')

    #用函数表示方法
    def qie(self):
        print('技能:切菜')

    def chao(self):
        print('技能:炒菜')
```

In [82]:
```
cook11=Cook5('小张','33岁')
```

```
__init__函数在实例化对象的时候会被自动执行
```

In [83]:
```
cook11.name1
```

Out[83]:
```
'小张'
```

定义一个计算器Calculator的类

类属性： 产地（Place_of_Origin） = ‘made in China'

实例属性： 颜色（color） 品牌（brand） 价格（price）

方法（技能）：

sum1(a, b)求两个数的和

sum2(a, b, c, d,....)求一系列数的和

sum3(a, b, c, d,...)求一系列数的倒数和

然后实例化一个对象出来试试功能

In [84]:
```
class Calculato:
    '这是一个计算机的类'
    #类属性
    Place_of_Origin='made in China'

    #实例属性
    def __init__(self,color,brand,price):
        self.color=color
        self.brand=brand
        self.price=price

    #类方法
    def sum1(self,a,b):
        return a+b

    def sum2(self,*args):
        s=0
        for i in args:
            s+=i
        return s

    def sum3(self,*args):
        s=0
        for i in args:
            s+=1/i
        return s
```

In [85]:
```
call=Calculato('黄色','晨光',20)
```

In [86]:
```
call.Place_of_Origin
```

Out[86]: 'made in China'

In [87]:
```
call.color
call.brand
call.price
```

Out[87]: '黄色'

Out[87]: '晨光'

Out[87]: 20

In [88]:
```
call.sum1(3,4)
call.sum2(1,2,3,4,5)
call.sum3(1,2,3)
```

Out[88]: 7

Out[88]: 15

Out[88]: 1.8333333333333333

2.10 删除类方法

In [90]:
```
del Calculato.sum1
```

In [95]:
```
call.sum1(1,2)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23336/2770072740.py in <module>
----> 1 call.sum1(1,2)

AttributeError: 'Calculato' object has no attribute 'sum1'
```

2.11 绑定类的方法

2.11.1 给类绑定方法

给类绑定的方法,所有从类中实例化出来的对象都可以使用

如果要删除绑定的方法,直接删除绑定的函数即可

比如刚才我们定义的计算器的类中，想再添加一个功能， sum4(a, b, c, d, ...)求一系列数平方和的函数

除了在类的定义中添加以外，还可通过types库的中的“MethodType”将自定义函数添加到类中

第一步：先有一个自定义好的函数，该函数的参数中，第一个位置上的参数一定得是self，self参数不参加函数的运算

第二步：导库from types import MethodType

第三步，绑定方法 类名字.方法名=MethodType（自定义好的函数的名字，类的名字）

In [98]:
```
def sum4(*args):
    s=0
    for i in args:
        s+=i**2
    return s
```

In [100]:
```
sum4(1,2,3)
```

Out[100]: 14

In [102]:
```
def sum4(self,*args):
    s=0
    for i in args:
        s+=i**2
    return s
```

In [103]:
```
sum4(1,2,3)  #因为对应位置接收的话,1被self接收了,所以args接收的是2和3,算的是2的平方+3的平方,输出13
```

Out[103]: 13

In [105]:
```
from types import MethodType  #从types模块中导入MethodType函数
```

In [106]:
```
Calculato.sum4=MethodType(sum4,Calculato)
#第一个参数是需要添加的方法函数,第二个参数是要给哪个类添加
```

In [107]:
```
cal2=Calculato('蓝色','卡西欧',50)
```

In [108]:
```
cal2.color
cal2.brand
cal2.price
```

Out[108]: '蓝色'

Out[108]: '卡西欧'

Out[108]: 50

In [109]:
```
cal2.sum4(1,2,3)
```

Out[109]: 14

In [112]:
```
#如果绑定的函数没有加上self参数,会报错,所以必须加上self参数
def sum5(*args):
    s=0
    for i in args:
        s+=i**2
    return s
```

In [113]:
```
Calculato.sum5=MethodType(sum5,Calculato)
```

In [114]:
```
cal3=Calculato('黑色','得力',250)
```

In [115]:
```
cal3.sum5(1,2,3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23336/413923035.py in <module>
----> 1 cal3.sum5(1,2,3)

~\AppData\Local\Temp/ipykernel_23336/725439063.py in sum5(*args)
      3     s=0
      4     for i in args:
----> 5         s+=i**2
      6     return s

TypeError: unsupported operand type(s) for ** or pow(): 'type' and 'int'
```

2.11.2 给实例(对象)绑定方法

给单独的实例绑定的方法只有这个实例能用,同类中实例出来的其他实例不能用

In [116]:
```
def sum1(self,a,b):
    return a+b
```

In [117]:
```
cal4=Calculato('紫色','得力',28)
```

In [118]:
```
cal4.sum1=MethodType(sum1,cal4)
#第一个参数写要绑定的方法函数,第二个对象写要给哪个对象进行绑定
```

In [119]:
```
cal4.sum1(1,2)
```

Out[119]: 3

In [120]:
```
cal3.sum1(2,3)  #单独给某个实例绑定了对象,只有这个实例可以调用,其他实例都不可以调用
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23336/3716853036.py in <module>
----> 1 cal3.sum1(2,3)

AttributeError: 'Calculato' object has no attribute 'sum1'
```

2.12 继承

继承是子类(派生类)继承父类(基类)的特征和行为,本质上是类与类之间的一种关系，子类能够继承父类 拥有的特征和技能，减少代码的冗余和工作量。

注意:在Python中，子类的实例都是父类的实例，但不能说父类的实例是子类的实例

定义一个关于人的类

In [124]:
```
class Person:
    """这是关于人的一个类"""
    language = "Y"#类属性,人类都有语言

    def __init__(self,color,race):#实例属性,肤色和种族
        self.color = color
        self.race = race

    def eat(self):#类方法
        print("技能:吃东西")
```

定义一个中国人的类

In [125]:
```
class ChinesePerson:
    """这是关于中国人的一个类"""
    language = "Y"
    country = "中国" #类属性,国籍

    def __init__(self,color,race,haircolor):#实例属性,肤色,种族和发色
        self.color = color
        self.race = race
        self.haircolor = haircolor

    def eat(self): #类方法
        print("技能:吃东西")

    def sowing(self): #类方法
        print("技能:播种")
```

这两个类各自定义都没有问题，但是，明显类ChinesePerson中很多东西类Person中也有，编程忌代码 重复出现太多，需要一种办法使类ChinesePerson可以直接用Person中的代码实现相同的功能,这就用 到了继承

2.12.1 继承的基本语法

继承的语法主要是在定义子类的代码中:

```
class 子类的名字（父类的名字）:

    子类的类属性

    子类的实例属性:
        父类名字.__init__(self,父类实例属性1，父类实例属性2.....）
        父类中没有的子类实例属性

    父类中没有的子类方法
```

父类

In [126]:
```
class Person:
    """这是关于人的一个类"""
    language = "Y"#类属性,人类都有语言

    def __init__(self,color,race):#实例属性,肤色和种族
        self.color = color
        self.race = race

    def eat(self):#类方法
        print("技能:吃东西")
```

子类

In [127]:
```
class ChinesePerson:
    """这是关于中国人的一个类"""
    language = "Y"
    country = "中国" #类属性,国籍

    def __init__(self,color,race,haircolor):#实例属性,肤色,种族和发色
        self.color = color
        self.race = race
        self.haircolor = haircolor

    def eat(self): #类方法
        print("技能:吃东西")

    def sowing(self): #类方法
        print("技能:播种
```

```
  File "C:\Users\99253\AppData\Local\Temp/ipykernel_23336/787580167.py", line 15
    print("技能:播种
                   ^
SyntaxError: EOL while scanning string literal
```

使用继承的语法定义中国人这个类

In [128]:
```
class ChinesePerson(Person):
    """这是关于中国人的一个类"""
    country = "中国" #类属性,国籍

    def __init__(self,color,race,haircolor):#实例属性,肤色,种族和发色
        self.haircolor = haircolor

    def sowing(self): #类方法
        print("技能:播种")
```

In [129]:
```
per=ChinesePerson('黄色','汉族','黑色')
```

In [130]:
```
per.language  #子类可以调用父类的属性
```

Out[130]: 'Y'

In [131]:
```
per.country
```

Out[131]: '中国'

In [132]:
```
per.color  #没有显示父类的实例属性
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23336/1316507668.py in <module>
----> 1 per.color  #没有显示父类的实例属性

AttributeError: 'ChinesePerson' object has no attribute 'color'
```

In [133]:
```
per.sowing()
```

```
技能:播种
```

In [134]:
```
per.eat()
```

技能:吃东西

In [143]:
```
class ChinesePerson(Person):
    """这是关于中国人的一个类"""
    country = "中国" #类属性,国籍

    def __init__(self,color,race,haircolor):#实例属性,肤色,种族和发色
        Person.__init__(self,color,race)  #必须加这行代码才会继承父类的实例属性  #self.color=color ,self.race=race
        self.haircolor = haircolor

    def sowing(self): #类方法
        print("技能:播种")
```

In [144]:
```
per1=ChinesePerson('黄色','苗族','黑色')
```

In [145]:
```
per1.language
```

Out[145]: 'Y'

In [146]:
```
per1.country
```

Out[146]: '中国'

In [150]:
```
per1.color
per1.race
per1.haircolor
```

Out[150]: '黄色'

Out[150]: '苗族'

Out[150]: '黑色'

In [148]:
```
per1.sowing()
```

技能:播种

In [149]:
```
per1.eat()
```

技能:吃东西

子类实例化的对象可以调用父类的类属性,父类实例化的对象不能调用子类特有的类属性

In [151]:
```
per2=Person('白色','英格兰')
```

In [152]:
```
per2.language
```

Out[152]: 'Y'

#父类实例化出来的对象不能调用子类特有的类属性和类方法

In [153]:
```
per2.country
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23336/398929349.py in <module>
----> 1 per2.country

AttributeError: 'Person' object has no attribute 'country'
```

In [154]:
```
per2.sowing()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23336/757787776.py in <module>
----> 1 per2.sowing()

AttributeError: 'Person' object has no attribute 'sowing'
```

2.12.2 方法重写

当父类的某个方法不完全适用于子类时，可以在子类里重新改写这个方法。此时子类和父类中都有这个函数，将优先使用子类中的

In [155]:
```
class Dog:
    legs='四条腿'

    def bark(self):
        print('汪汪')
```

In [156]:
```
dog1=Dog()
```

In [157]:
```
dog1.legs
```

Out[157]: '四条腿'

In [158]:
```
dog1.bark()
```

```
汪汪
```

In [ ]:
```

```

其他子类的狗狗叫声都是汪汪,哈士奇的叫声和别的狗狗不一样,需要重写哈士奇这一个子类的叫声

In [159]:
```
class Husky(Dog):
    def bark(self):
        print('嗷嗷')
```

In [160]:
```
erha=Husky()
```

In [161]:
```
erha.legs
```

Out[161]: '四条腿'

In [162]:
```
erha.bark()  #子类和父类有相同名字的方法时,从子类实例化出来的对象优先使用子类的方法
```

嗷嗷

2.13 多态

不同的对象调用相同的方法得到不同的结果

其实就是继承+方法重写就可以实现多态。

In [163]:
```
class Animal:
    def bark(self):
        print('*^($^$)$%$&')
```

In [164]:
```
class Dog(Animal):
    def bark(self):
        print('汪汪')
```

In [165]:
```
class Cat(Animal):
    def bark(self):
        print('喵喵')
```

In [166]:
```
class Sheep(Animal):
    def bark(self):
        print('咩咩')
```

In [167]:
```
dog=Dog()
cat=Cat()
sheep=Sheep()
```

In [168]:
```
dog.bark()
cat.bark()
sheep.bark()  #不同对象调用相同的方法结果不一样,就是多态
```

```
汪汪
喵喵
咩咩
```

-什么是类?
-什么是对象?实例化是什么?
-什么是属性?
-什么是方法?
-通过对象调属性和方法
-实例化后对象的属性可以进行修改