**类与对象**

**类和对象**

对象一个具体的事物，类是具有相同特征和功能的对象的抽象。对象是类的实例

**定义类**

class 类名(首字母大写)：

成员变量

成员函数

class Fruit:  
 color = "orange"  
 price = 4  
 def f(self):  
 return 'good'

**实例化类，创建对象**

orange = Fruit()

对象可以调用类中的属性和方法，类中的self代表实例本身，其地址指针指向实例的内存地址

print(orange.color)

print(orange.f())

>>>

Orange

good

**类的属性**

可按调用对象分为类属性和实例属性，也可按可调用范围分为公有属性和私有属性，此外还有内置属性

**一、类属性**

类和实例都可以调用的属性，用于记录某一项需要保持不变的数据，实例不能对类属性进行修改

class Fruit:  
 belong = 'plant'*#类属性*orange = Fruit('orange')

**二、实例属性**

只有实例才可以调用的属性，一般在初始化方法中定义，但也可以在其他方法中定义

class Fruit:

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

self.color = color*#实例属性*

def getPrice(self):

self.price = 10*#实例属性*

orange = Fruit('orange')

orange.getPrice()

print(orange.price)

>>> 10

**三、公有属性**

属性名，公有属性可以直接通过类调用，不一定需要实例

class Fruit:  
 color = "orange"  
 price = 4

print(Fruit.color)

>>>orange

**四、私有属性**

\_\_属性名，私有属性的特点是不允许在类外调用或更改

class Fruit:  
 color = "orange"  
 price = 4  
 \_\_cost = 3

print(orange.\_\_cost)

>>>报错

class Fruit:  
 color = "yellow"  
 price = 4  
 \_\_cost = 3  
 def say(self):  
 print(self.\_\_cost)  
orange = Fruit()  
orange.say()

>>>3

也可以为实例定义私有属性

class Fruit:  
 color = "orange"  
 price = 4  
 def \_\_init\_\_(self):  
 self.size = 'big'  
 self.\_\_cost = 3  
 flag = 1 *#函数体内部的局部变量，作用域只在\_\_init\_\_函数内部*orange = Fruit()  
print(orange.\_\_cost)

>>>报错

class Fruit:  
 color = "orange"  
 price = 4  
 def \_\_init\_\_(self):  
 self.size = 'big'  
 self.\_\_cost = 3  
 flag = 1 *#函数体内部的局部变量，作用域只在\_\_init\_\_函数内部* def printf(self):  
 print(self.\_\_cost)  
orange = Fruit()  
orange.printf()

>>>3

**五、内置属性**

\_\_属性名\_\_，内置属性无需定义，由python内部定义，内置属性一般是公有的

Class.\_\_name\_\_

以字符串形式返回类名

print(A.\_\_name\_\_)

>>> 'A'

instance.\_\_doc\_\_

返回类的注释

class Fruit:  
 *"""Fruit Description"""*

color = "orange"  
 price = 4  
print(Fruit.\_\_doc\_\_)*#打印类的注释*

>>> Fruit Description

instance.\_\_class\_\_

返回实例所属的类，相当于type(实例)

class A():

pass

a = A()

print(type(a))

print(a.\_\_class\_\_)

>>>

<class '\_\_main\_\_.A'>

<class '\_\_main\_\_.A'>

instance.\_\_dict\_\_

使用类的实例对象调用 \_\_dict\_\_，会输出由类中所有实例属性组成的字典

class CLanguage:

a = 1

b = 2

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

self.name = "C语言中文网"

self.add = "http://c.biancheng.net"

#通过类实例对象调用 \_\_dict\_\_

clangs = CLanguage()

print(clangs.\_\_dict\_\_)

>>>'name': 'C语言中文网', 'add': 'http://c.biancheng.net'}

**类的方法**

**一、公有方法**

方法名

**二、私有方法**

\_\_方法名，私有方法只能通过公有方法在类内部调用

class Fruit:  
 color = "yellow"  
 price = 4  
 \_\_cost = 3  
 def say(self):  
 print(self.\_\_cost)  
 def \_\_getCost(self):  
 print(self.\_\_cost)  
orange = Fruit()  
orange.\_\_getCost()

>>>报错

class Fruit:  
 color = "yellow"  
 price = 4  
 \_\_cost = 3  
 def say(self):  
 print(self.\_\_cost)  
 self.\_\_getCost()  
 def \_\_getCost(self):  
 print(self.\_\_cost)  
orange = Fruit()  
orange.say()

>>>

3

2

**三、魔法方法**

\_\_方法名\_\_，这些方法在类或对象的特定事件发生后会自动执行，如果希望根据自己的程序定制特殊功能的类，那么就需要对这些方法进行重写，魔法方法的“\_\_方法名\_\_”作为整体被Python识别，不能改变魔法方法的名称，否则魔法方法将失效。实际上，存在一个更原始的方法或函数，可以判断特定事件是否发生，并据此调用魔法方法。

**常用的魔法方法**

**\_\_init\_\_()**

在实例化类时自动执行，用于初始化实例属性

class Human:  
 def \_\_init\_\_(self,name,gender,age):   
 self.name = name  
 self.gender = gender  
 self.age = age  
LiMing = Human('LiMing','Man',21)

**\_\_call\_\_()**

将实例对象像普通函数一样，以“对象名()”的形式调用。实际上，“对象名()”可理解为“对象名.\_\_call\_\_()”的简写

class CLanguage:  
 *# 定义\_\_call\_\_方法* def \_\_call\_\_(self,name,add):  
 print(name,add)  
clangs = CLanguage()  
clangs("C语言中文网","http://c.biancheng.net")

>>>C语言中文网 http://c.biancheng.net

**\_\_add\_\_()**

出现“+”自动调用

class A:  
 def \_\_init\_\_(self,x):  
 self.x = x  
 def \_\_add\_\_(self,other):

print(1)  
 return int(self.x)+int(other.x)  
a = A(3.3)  
b = A(5.2)  
print(a,b)  
print(a+b)

>>>

<\_\_main\_\_.A object at 0x000002A86FE28400> <\_\_main\_\_.A object at 0x000002A8702774C0>

1

8

**\_\_str\_\_()**

返回值是str类型的，当我们需要以字符串的形式输出对象时(如调用print时)，就会自动调用该方法

class A:  
 def \_\_str\_\_(self):  
 return '我真帅'  
a = A()  
print(a)

>>> 我真帅

**\_\_getitem\_\_()**

instance[key]相当于调用instance.\_\_getitem\_\_(key)

**\_\_len\_\_()**

len(instance)相当于调用instance.\_\_len\_\_()

**类的继承**

子类继承父类的公有属性，公有方法，不能继承父类的私有属性，私有方法

class Fruit:  
 belong = 'plant'  
 def \_\_init\_\_(self,color,size,taste,price):  
 self.color = color  
 self.size = size  
 self.taste = taste  
 self.price = price  
 def getPrice(self):  
 return self.price  
class Orange(Fruit):  
 origin = 'China'*#子类可以定义自己的类属性* def getColor(self):*#子类可以定义自己的方法* return self.color  
orange = Orange('orange','big','sweet and sour',5)*#子类继承了父类的构造函数*  
print(orange.color)  
print(orange.origin)  
print(orange.getPrice())*#子类继承了父类的公有(一般)方法*print(orange.getColor())

>>>

orange

China

5

orange

**super()**

如果子类和父类都定义了同名方法，那么子类的方法就会覆盖父类的，此时调用父类的方法需调用Python的内置函数super()

在Python3中可直接用super().xxx，同时也兼容Python2中的super(SubClass\_name, self).xxx

class A:  
 def add(self,x):  
 y = x+1  
 print(y)  
class B(A):  
 def add(self,x):  
 print(1)  
 super().add(x)  
b = B()  
b.add(1)

>>>

1

2

class Person:  
 def \_\_init\_\_(self,name='Person'):  
 self.name = name  
class Man(Person):  
 def \_\_init\_\_(self,name,PEscore):  
 super(Man,self).\_\_init\_\_(name) *#super().\_\_init\_\_()使子类在加入新属性的同时继承父类属性* self.PEscore = PEscore  
LiMing = Man('LiMing',90)  
print(LiMing.name)  
print(LiMing.PEscore)

>>>

LiMing

90

**方法重载**

重载即在子类中重写父类方法

class Fruit:  
 belong = 'plant'  
 def \_\_init\_\_(self,color,size,taste,price):  
 self.color = color  
 self.size = size  
 self.taste = taste  
 self.price = price  
 def getPrice(self):  
 return self.price  
class Orange(Fruit):  
 def getPrice(self):*#重写父类中的getPrice方法* return "the orange costs %d yuan" %(self.price)  
orange = Orange('orange','big','sweet and sour',5)  
print(orange.getPrice())

>>> the orange costs 5 yuan

如果仍然想用父类方法，可以在子类中的任何地方调用父类方法

class Fruit:  
 def \_\_init\_\_(self,price):  
 self.price = price  
 def getPrice(self):  
 return self.price  
class Orange(Fruit):  
 def getCost(self):  
 return Fruit.getPrice(self)-1*#调用父类方法*  
orange = Orange(5)  
print(orange.getCost())

>>>4