第8章 Python与面向对象（二）

AI程序设计课程组

目录页

抽象类

抽象类

- 在面向对象中，如果我们想要为多个实体类定义一个父类，但是我们不想让父类直接实例化，因为父类的实例化没有意义（如下案例，动物的叫声）。

```
class Animal(object):
    def __init__(self, name):
        self.name = name

    def makesound(self):
        pass

    def sayname(self):
        print(self.name)
```

```
class cat(Animal):
    def makesound(self):
        print(self.name,'is Meowing!')
class dog(Animal):
    pass
```

抽象类

- 我们想要为多个实体类定义一个父类，但是我们不想让父类直接实例化，因为父类的实例化没有意义。

```
class Animal(object):
    def __init__(self, name):
        self.name = name

    def makesound(self):
        pass

    def sayname(self):
        print(self.name)
```

```
class cat(Animal):
    def makesound(self):
        print(self.name,'is Meowing!')
class dog(Animal):
    pass
```

此处dog类并没有实现makesound方法，导致父类方法失效，因此需要更强的约束

抽象类

- 我们需要定义一个类，但是这个类本身并不需要实例化，而只是作为一种模板或规范存在。这种类就叫做抽象类。在Python中，我们可以通过ABC模块来定义抽象类和接口。

```
from abc import ABC, abstractmethod
class Animal(ABC):
    def __init__(self, name):
        self.name = name

    @abstractmethod
    def make_sound(self):
        pass

    def sayname(self):
        print(self.name)
```

```
class dog(Animal):
    pass
dog1=dog('Snoopy')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-71dbbfd66054> in <module>
      1 class dog(Animal):
      2     pass
----> 3 dog1=dog('Snoopy')

TypeError: Can't instantiate abstract class dog with abstract methods make_sound
```

ABC模块保证抽象类的方法必须被子类实现

抽象类

- ABC模块保证抽象类的方法必须被子类实现

```
from abc import ABC, abstractmethod
class Animal(ABC):
    def __init__(self, name):
        self.name = name

    @abstractmethod
    def make_sound(self):
        pass

    def sayname(self):
        print(self.name)
```

```
class dog(Animal):
    def __init__(self,name):
        self.name=name
    def make_sound(self):
        print(self.name,'is barking!')

class cat(Animal):
    def __init__(self,name):
        self.name=name
    def make_sound(self):
        print(self.name,'is Meowing!')
```

```
dog1=dog('snoopy')
dog1.make_sound()
```

```
snoopy is barking!
```

```
cat1=cat('garfield')
cat1.make_sound()
```

```
garfield is Meowing!
```

运算符重载

运算符重载

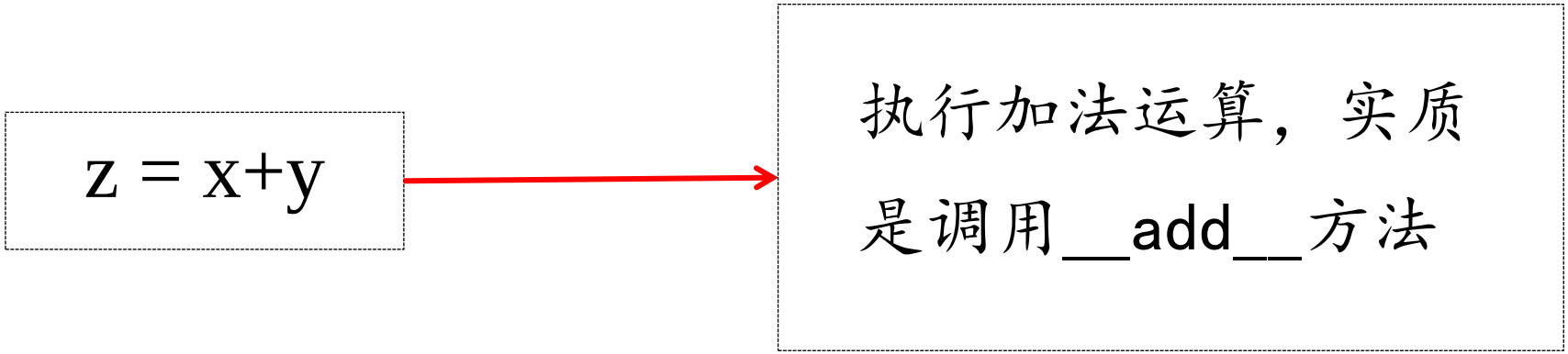
- 运算符重载是通过实现特定的方法使类的实例对象支持Python的各种内置操作。例如：+运算符是类里提供的__add__这个函数，当调用+实现加法运算的时候，实际上是调用了__add__方法。

运算符重载

方法	说明	何时调用方法
__add__	加法运算	对象加法：x+y，x+=y
__sub__	减法运算	对象减法：x-y，x-=y
__mul__	乘法运算	对象乘法：x*y，x*=y
__div__	除法运算	对象除法：x/y，x/=y
__getitem__	索引，分片	x[i]、x[i:j]、没有__iter__的for循环等
__setitem__	索引赋值	x[i]=值、x[i:j]=序列对象
__delitem__	索引和分片删除	del x[i]、del x[i:j]

运算符重载

- 加法运算是通过调用__add__方法完成重载的，当两个实例对象执行加法运算时，自动调用__add__方法。



算术运算符重载案例“+”


```
class Vector:
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def __add__(self,other):
        if isinstance(other, Vector):
            return Vector(self.x+other.x,self.y+other.y)
        else:
            print('Invalid operation')
    def __str__(self):
        return 'The Vector is({},{})'.format(self.x,self.y)
```

```
vector1=Vector(1,2)
vector2=Vector(5,6)
result=vector1+vector2
print(result)
```

算术运算符重载案例“*”


```
class Matrix:
    def __init__(self,rows,cols):
        self.rows=rows
        self.cols=cols
        self.data=[[0]*cols for _ in range(rows)]

    def __mul__(self,scalar):
        if isinstance(scalar,int):
            result=Matrix(self.rows,self.cols)
            for i in range(self.rows):
                for j in range(self.cols):
                    result.data[i][j]=self.data[i][j]*scalar
            return result
        else:
            print('无效操作')

    def __str__(self):
        return str(self.data)
```

```
tempmatrix=Matrix(2,2)
tempmatrix.data=[[1,2],[3,4]]
result=tempmatrix*2
print(result)
```

比较运算符重载案例“==”


```
#创建Person类
class Person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
    #重载运算符“==”
    def __eq__(self,other):
        if isinstance(other,Person):
            return self.name==other.name and self.age==other.age
        else:
            return 0
    #重载运算符“!=”
    def __ne__(self,other):
        return not self.__eq__(other)
```

```
zs=Person('Zhangsan',30)
ls=Person('Lisi',28)
ww=Person('Wangwu',27)
test=Person('Wangwu',27)

print(zs==ls)
print(ls==ww)
print(ww==test)
print(zs!=ls)
```

```
False
False
True
True
```

索引和分片重载案例

跟索引相关的重载方法包括如下3个：

- __getitem__：索引、分片；
- __setitem__：索引赋值；
- __delitem__：索引和分片删除。

索引和分片重载

1. __getitem__方法

在对实例对象执行索引、分片或者for迭代操作时，会自动调用__getitem__方法。

```
# 定义索引、分片运算符重载方法
    def __getitem__(self, index):
        return self.data[index]
```

2. __setitem__方法

通过赋值语句给索引或者分片赋值时，调用__ setitem __方法实现对序列对象的修改。

```
def __setitem__(self, index, value):
    self.data[index] = value
```

3. __delitem__方法

当调用del方法时，实质上会调用__delitem__方法实现删除操作。

```
def __delitem__(self, index):
    del self.data[index]
```

```
class Score:
    def __init__(self,name,data):
        name=name
        self.data= data
    def __getitem__(self, index):
        return self.data[index]
    def __setitem__(self, index, value):
        self.data[index] = value
```

```
zs=Score('zhangsan',[90,89,99])
print(zs[0])
```

```
90
```

类与对象实例

实例1

创建类，判断同学们成绩的水平，60-80为中等，80-100为优秀，0-60为不合格

```
#类方法的定义及调用
class Stu:
    def __init__(self,name,score):
        self.name=name
        self.score=score
    def getgrade(self):
        if self.score>=60 and self.score<=80:
            return '您的成绩为中等'
        elif 80<self.score<=100:
            return '您的成绩为优秀'
        elif self.score>100:
            return '分数不合法'
        else:
            return '您的成绩不合格'

if __name__=='__main__':
    tempstu=Stu('zhangsan',80)
    print(tempstu.getgrade())
    print(tempstu)
```

```
您的成绩为中等
<__main__.Stu object at 0x0000020D267C1B00>
```

此处如果让你重写Stu方法的输出，实现：print(tempstu) 即输出“zhangsan的成绩为良好”该如何实现？

实例2

创建学生类，包含学生学号、姓名和学历情况

```
class Stu:
    def __init__(self,id,name):
        self.id=id
        self.name=name
        self.edu=[]
    def addEdu(self,edu):
        self.edu.append(edu)
class Edu:
    def __init__(self,name):
        self.name=name

if __name__=='__main__':
    s=Stu(10,'zs')
    for temp in ('本科 硕士 博士').split(' '):
        e=Edu(temp)
        s.addEdu(e)

for j in s.edu:
    print(j.name)
```

本科
硕士
博士

类的属性为另一个类的对象

实例3

创建汽车类，包含引擎、车轮等信息

```
class Engine:
    def __init__(self,power):
        self.power=power
```

```
class Tire:
    def __init__(self,size):
        self.size=size
```

```
class Car:
    def __init__(self,engine,tires):
        self.engine=engine
        self.tires=tires
```

```
engine=Engine(200)
tires=[Tire(60),Tire(16),Tire(16),Tire(16)]
car=Car(engine,tires)
```

“汽车”是由“发动机”、“轮胎”等部件组成的，因此构建了汽车类，汽车类由轮胎和发动机类的实例组成。这是一种组合关系。

如果描述汽车上有很多乘客，请创建乘客类，并实现

实例4

构建电话本类，能够实现按照姓名进行模糊查找。例如查找“张”，能找到所有名字里包含“张”的人的电话本数据

```
class ContactList(list):
    def match(self,name):
        matchResult=[]
        for contact in self:
            if name in contact.name:
                matchResult.append(contact)
        return matchResult

class Contact:
    allContacts=ContactList()
    def __init__(self,name,email):
        self.name=name
        self.email=email
        Contact.allContacts.append(self)
```

创建了一个新的ContactList类来扩展Python的内置list。而不是实例化一个普通的列表作为类变量。

本章小结

- 类与对象
- 类的定义与使用
- 封装
- 继承
- 多态
- 抽象类
- 运算符重载