继承

语法角度讲

继承方法

1. 代码:
   class 父类:
   def 父类方法(self):
   方法体

   class 子类(父类):
   def 子类方法(self):
   方法体

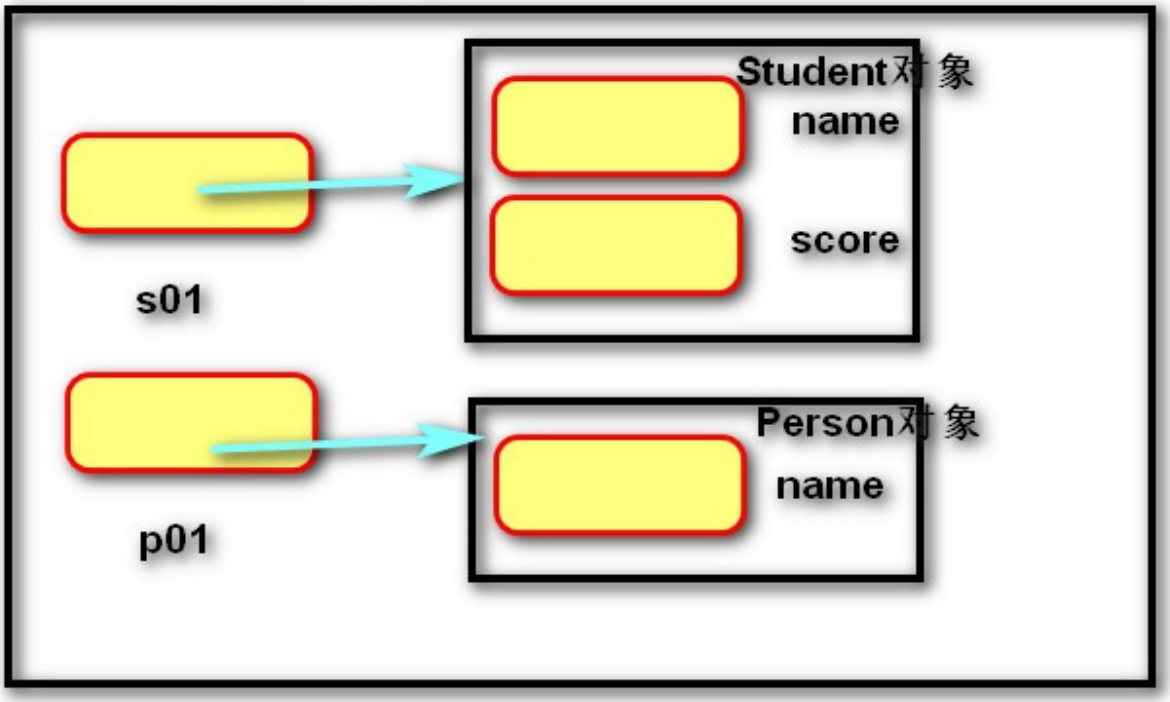
   儿子 = 子类()
   儿子.子类方法()
   儿子.父类方法()

2. 说明:
   子类直接拥有父类的方法.

```
"""
    继承 -- 数据
"""
class Person:
    def __init__(self, name=""):
        self.name = name
class Student(Person):
    def __init__(self, name="", score=0):
        self.score = score
        # self.name = name
        # 如果子类没有构造函数,使用父类构造函数
        # 如果子类有构造函数,必须通过super()调用父类构造函数,
        # 否则会覆盖父类(不执行)的.
        super().__init__(name)
# 创建实例变量
s01 = Student("无忌", 100)
print(s01.name)
```

```
print(s01.score)
p01 = Person("翠山")
print(p01.name)
```

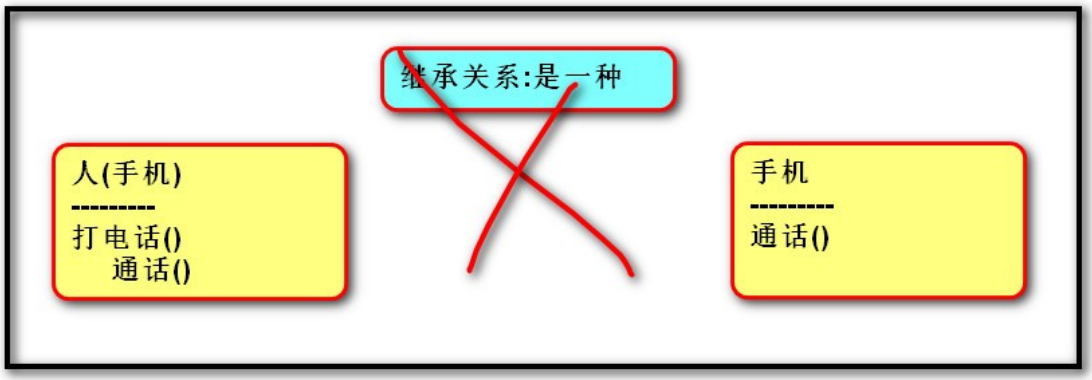
```
class Person:
    def __init__(self, name=""):
        self.name = name

class Student(Person):
    def __init__(self,name = "",score=0):
        self.score = score
        super().__init__(name)

# 创建实例变量
s01 = Student("无忌",100)
p01 = Person("翠山")
```

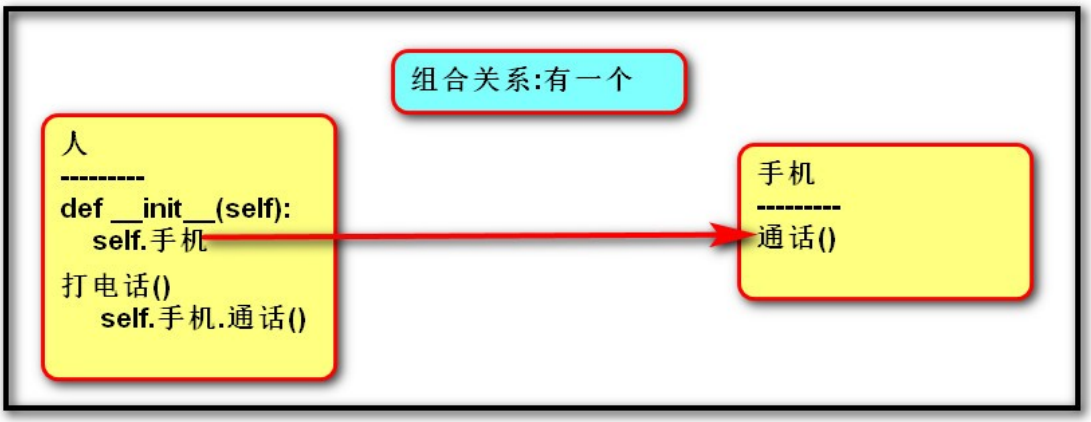


内置函数

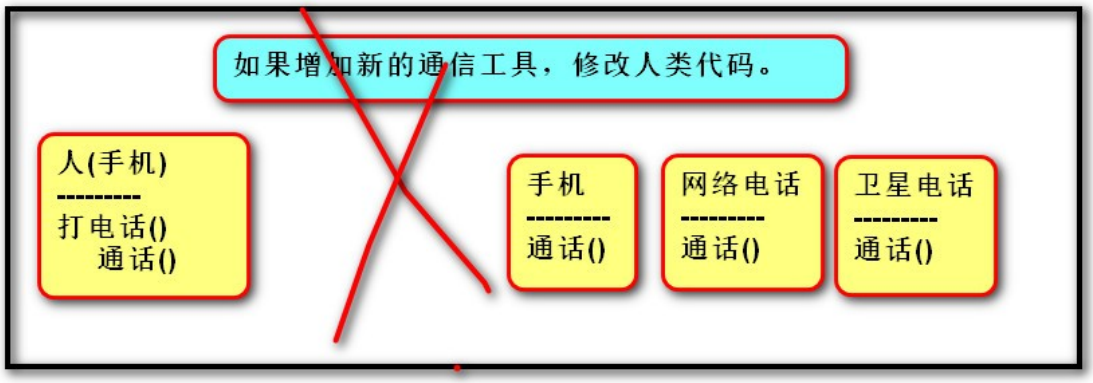
isinstance(对象, 类型)
返回指定对象是否是某个类的对象。
issubclass(类型，类型)
返回指定类型是否属于某个类型。
"""

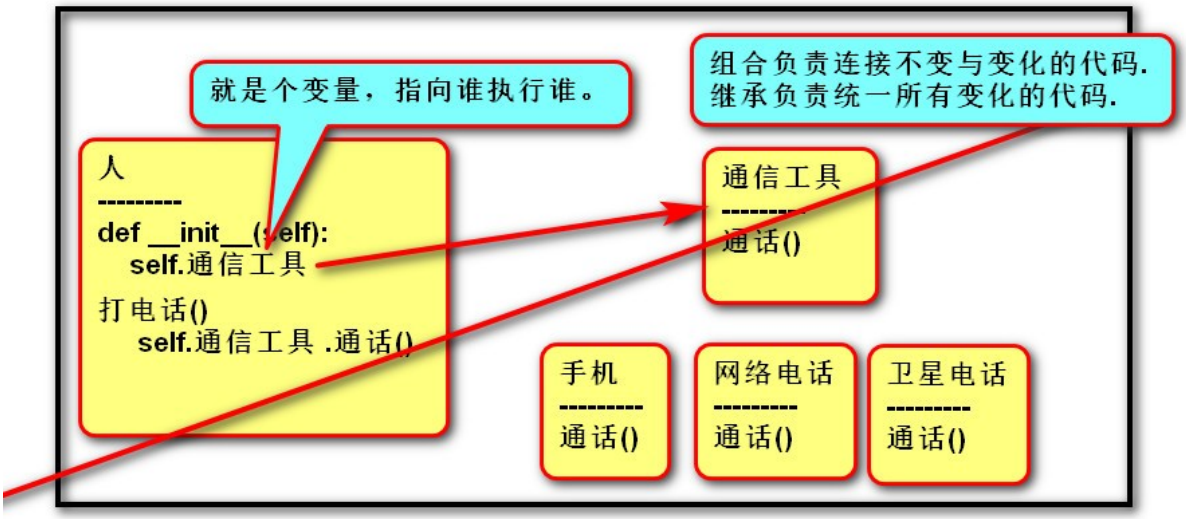
```
   继承 -- 方法
      财产:钱不用儿子挣,但是可以花.
      皇位:江山不用太子打,但是可以坐.
      编程:代码不用子类写,但是可以用.
"""
class Person:
   def say(self):
      print("说话")
class Student(Person):
   def study(self):
      print("学习")
class Teacher(Person):
   def teach(self):
      print("讲课")
# 创建父类型对象,只能访问父类型成员
p01 = Person()
p01.say()
# 创建子类型对象,能访问父类型成员,还能访自身成员
s01 = Student()
s01.say()
s01.study()
# "是不是 实例"
# s01的对象 是 一种 Student类型
print(isinstance(s01,Student))# True
print(isinstance(s01,Person))# True
print(isinstance(s01,Teacher))# False
# s01的类型 等于 Student类型
print(type(s01) == Student)# True
print(type(s01) == Person)# False
# Student 类型 是一种 Person类型
print(issubclass(Student,Person))# True
print(issubclass(Student,Student))# True
print(issubclass(Student,Teacher))# False
# 练习:定义父类(动物,行为--吃)
#     定义子类(狗,行为-- 跑)
#     定义子类(鸟,行为-- 飞)
#     分别创建父类子类对象,调用其行为.
#     体会isinstance  type  issubclass三者之间的关系
```





需求：人还可能使用座机/网络电话/卫星电话....





继承数据

1. 代码

```
class 子类(父类):
    def __init__(self,参数列表):
        super().__init__(参数列表)
        self.自身实例变量 = 参数
```

2. 说明

子类如果没有构造函数，将自动执行父类的，但如果有构造函数将覆盖父类的。此时必须通过super()函数调用父类的构造函数，以确保父类实例变量被正常创建。

定义

重用现有类的功能，并在此基础上进行扩展。
说明：子类直接具有父类的成员（共性），还可以扩展新功能。

优点

一种代码复用的方式。

缺点

耦合度高：父类的变化，直接影响子类。

设计角度讲

定义

将相关类的共性进行抽象，统一概念，隔离变化。

适用性

多个类在概念上是一致的，且需要进行统一的处理。

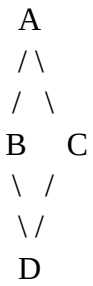
相关概念

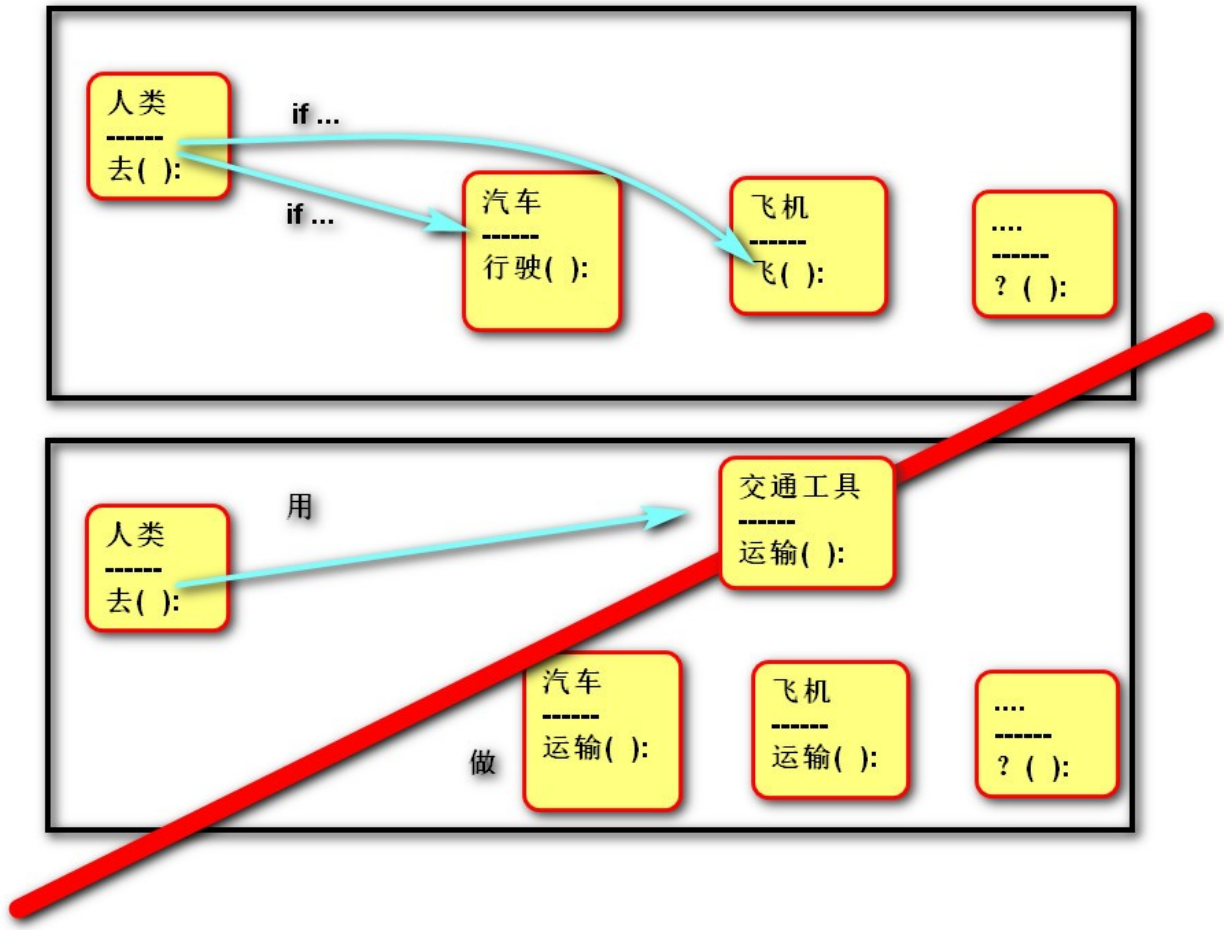
父类（基类、超类）、子类（派生类）。
父类相对于子类更抽象，范围更宽泛；子类相对于父类更具体，范围更狭小。

单继承：父类只有一个（例如 Java，C#）。
多继承：父类有多个（例如 C++，Python）。
Object 类：任何类都直接或间接继承自 object 类。

多继承

一个子类继承两个或两个以上的基类，父类中的属性和方法同时被子类继承下来。
同名方法的解析顺序（MRO， Method Resolution Order）：
类自身 --> 父类继承列表（由左至右）--> 再上层父类






```
"""
    继承 -- 设计(1)
"""
# 老张开车去东北.
# 需求变化:还可能坐飞机,坐火车.....
# 违背了面向对象设计原则:
#    开闭原则 /  依赖倒置
class Person:
    def __init__(self, name=""):
        self.name = name
    # 对方法增加语句,删除语句,修改语句都视为改变代码.
    def go_to(self,str_position,vehicle):
        print("去:",str_position)
        if type(vehicle) == Car:
            vehicle.run()
        elif type(vehicle) == Airplane:
            vehicle.flay()
class Car:
```

```
    def run(self):
        print("走你....")
class Airplane:
    def flay(self):
        print("嗖嗖...")
c01 = Car()
a01 = Airplane()
lz = Person("老张")
lz.go_to("东北",c01)
```

"""

```
    继承 -- 设计(2)
    练习:exercise02
"""
# 老张开车去东北.
# 需求变化:还可能坐飞机,坐火车.....
class Person:
    def __init__(self, name=""):
        self.name = name
    def go_to(self,str_position,vehicle):
        print("去:",str_position)
        vehicle.transport()
class Vehicle:
    """
        抽象的
        交通工具
    """
    def transport(self):
        pass
# ------------------------------------
class Car(Vehicle):
    def transport(self):
        print("走你...")
class Airplane(Vehicle):
    def transport(self):
        print("嗖嗖嗖.")
c01 = Car()
a01 = Airplane()
lz = Person("老张")
lz.go_to("东北",a01)
```

多态

设计角度讲

定义

父类的同一种动作或者行为，在不同的子类上有不同的实现。

作用

1. 在继承的基础上，体现类型的个性化（一个行为有不同的实现）。
2. 增强程序扩展性，体现开闭原则。

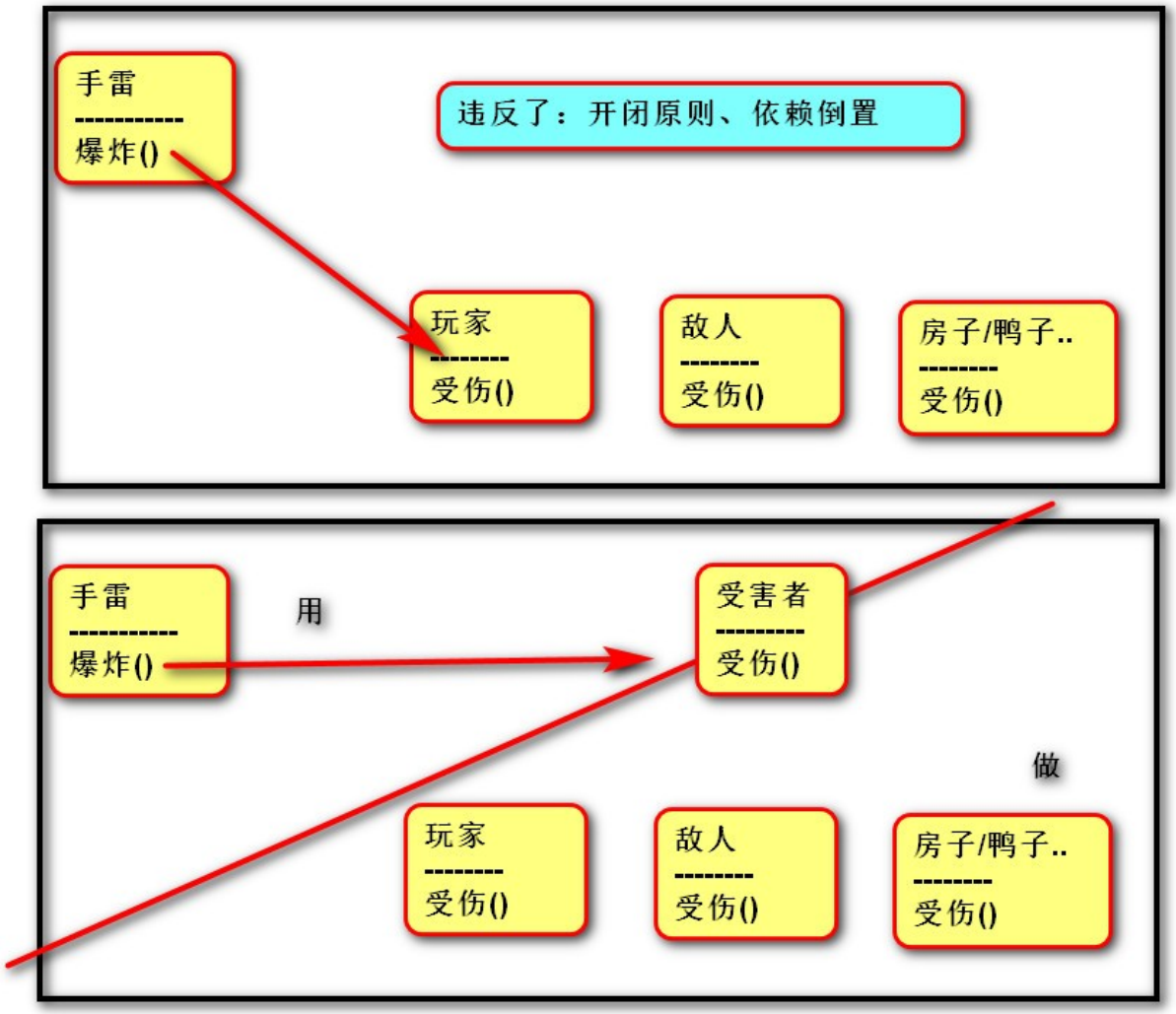
语法角度讲

重写

子类实现了父类中相同的方法（方法名、参数）。
在调用该方法时，实际执行的是子类的方法。

快捷键

Ctrl + O




```
"""
    练习:手雷爆炸了,可能伤害敌人/玩家的生命.
        需求变化:房子/鸭子...
"""
#----------------架构师---------------
class Granade:
    """
        手雷
    """
    def __init__(self,atk):
        self.atk = atk
    def explode(self,sufferer):
```

```
        print("爆炸啦")
        # 调用的是父类受伤方法
        # 执行的是子类受伤方法
        sufferer.damage(self.atk)
class Sufferer:
    """
        受害者
    """
    def damage(self,value):
        pass
# ---------------初/中级程序员-----------------
class Player(Sufferer):
    def damage(self, value):
        print("玩家受伤啦")
class Enemy(Sufferer):
    def damage(self, value):
        print("敌人受伤啦")
# ------------------测试-----------------------
g01 = Granade(50)
p01 = Player()
e01 = Enemy()
g01.explode(p01)
```

```
# 定义:图形管理器
#         1. 记录所有图形
#         2. 提供计算总面积的方法
# 需求变化:
#         圆形(pi * r ** 2)
#         矩形(长 * 宽)
#         ...
# 测试:
# 创建图形管理器,添加圆形,矩形.
# 调用计算总面积方法.
class GraphicManager:
    """
        图形管理器
    """
    def __init__(self):
        # 存储的是具体图形
        # 所以图形管理器与图形是组合关系.
        self.__list_graphic = []
```

```
    @property
    def list_graphic(self):
        return self.__list_graphic
    def add_graphic(self, graphic):
        self.__list_graphic.append(graphic)
    def get_total_area(self):
        total_area = 0
        for item in self.__list_graphic:
            # 调用父类,执行子类.
            total_area += item.calculate_area()
        return total_area
class Graphic:
    """
        图形,抽象所有具体的图形,统一图形的计算面积方法,隔离具体图形的变化.
    """
    def calculate_area(self):
        """
            计算面积
        :return: 小数类型的面积.
        """
        pass
# ----------------------------------------
class Circle(Graphic):
    def __init__(self, r):
        self.r = r
    def calculate_area(self):
        return 3.14 * self.r ** 2
class Rectanle(Graphic):
    def __init__(self, lenght, width):
        self.length = lenght
        self.width = width
    def calculate_area(self):
        return self.length * self.width
# ------------
manager = GraphicManager()
c01 = Circle(5)
manager.add_graphic(c01)
manager.add_graphic(Rectanle(2, 5))
print(manager.get_total_area())
```

内置可重写函数

Python 中，以双下划线开头、双下划线结尾的是系统定义的成员。我们可以在自定义类中进行重写，从而改变其行为。

转换字符串

__str__函数：将对象转换为字符串(对人友好的)
__repr__函数：将对象转换为字符串(解释器可识别的)

运算符重载

定义：让自定义的类生成的对象(实例)能够使用运算符进行操作。

算数运算符

方法名	运算符和表达式	说明
`__add__(self, rhs)`	self + rhs	加法
`__sub__(self, rhs)`	self - rhs	减法
`__mul__(self, rhs)`	self * rhs	乘法
`__truediv__(self, rhs)`	self / rhs	除法
`__floordiv__(self, rhs)`	self // rhs	地板除
`__mod__(self, rhs)`	self % rhs	取模(求余)
`__pow__(self, rhs)`	self ** rhs	幂

反向算数运算符重载

方法名	运算符和表达式	说明
`__radd__(self, lhs)`	lhs + self	加法
`__rsub__(self, lhs)`	lhs - self	减法
`__rmul__(self, lhs)`	lhs * self	乘法
`__rtruediv__(self, lhs)`	lhs / self	除法
`__rfloordiv__(self, lhs)`	lhs // self	地板除
`__rmod__(self, lhs)`	lhs % self	取模(求余)
`__rpow__(self, lhs)`	lhs ** self	幂

复合运算符重载

方法名	运算符和复合赋值语句	说明
`__iadd__(self, rhs)`	self += rhs	加法
`__isub__(self, rhs)`	self -= rhs	减法
`__imul__(self, rhs)`	self *= rhs	乘法
`__itruediv__(self, rhs)`	self /= rhs	除法
`__ifloordiv__(self, rhs)`	self //= rhs	地板除
`__imod__(self, rhs)`	self %= rhs	取模(求余)
`__ipow__(self, rhs)`	self **= rhs	幂

比较运算重载

方法名	运算符和复合赋值语句	说明
`__lt__(self, rhs)`	self < rhs	小于
`__le__(self, rhs)`	self <= rhs	小于等于
`__gt__(self, rhs)`	self > rhs	大于
`__ge__(self, rhs)`	self >= rhs	大于等于
`__eq__(self, rhs)`	self == rhs	等于
`__ne__(self, rhs)`	self != rhs	不等于

设计原则

开-闭原则（目标、总的指导思想）

Open **C**losed **P**rinciple
对扩展开放，对修改关闭。
增加新功能，不改变原有代码。

类的单一职责（一个类的定义）

Single **R**esponsibility **P**rinciple
一个类有且只有一个改变它的原因。

依赖倒置（依赖抽象）

Dependency **I**nversion **P**rinciple
客户端代码(调用的类)尽量依赖(使用)抽象。
抽象不应该依赖细节，细节应该依赖抽象。