魔术方法 ***

实例化

方法	意义
`__new__`	实例化一个对象 该方法需要返回一个值，如果该值不是cls的实例，则不会调用`__init__` 该方法永远都是静态方法

```
class A:
    def __new__(cls, *args, **kwargs):
        print(cls)
        print(args)
        print(kwargs)
        #return super().__new__(cls)
        #return 1
        return None

    def __init__(self, name):
        self.name = name

a = A()
print(a)
```

`__new__`方法很少使用，即使创建了该方法，也会使用`return super().__new__(cls)`基类object的`__new__`方法来创建实例并返回。

可视化

方法	意义
`__str__`	str()函数、format()函数、print()函数调用，需要返回对象的字符串表达。如果没有定义，就去调用`__repr__`方法返回字符串表达，如果`__repr__`没有定义，就直接返回对象的内存地址信息
`__repr__`	内建函数repr()对一个对象获取**字符串**表达。 调用`__repr__`方法返回字符串表达，如果`__repr__`也没有定义，就直接返回object的定义就是显示内存地址信息
`__bytes__`	bytes()函数调用，返回一个对象的bytes表达，即返回bytes对象

```
class A:
    def __init__(self, name, age=18):
        self.name = name
        self.age = age

    def __repr__(self):
        return 'repr: {},{}'.format(self.name, self.age)
```

```
    def __str__(self):
        return 'str:  {},{}'.format(self.name, self.age)

    def __bytes__(self):
        #return "{} is {}".format(self.name, self.age).encode()
        import json
        return json.dumps(self.__dict__).encode()


print(A('tom')) # print函数使用__str__
print('{}'.format(A('tom')))
print([A('tom')]) # []使用__str__，但其内部使用__repr__
print([str(A('tom'))]) # []使用__str__，其中的元素使用str()函数也调用__str__
print(bytes(A('tom')))
```

bool

方法	意义
`__bool__`	内建函数bool()，或者对象放在逻辑表达式的位置，调用这个函数返回布尔值。 没有定义`__bool__`()，就找`__len__`()返回长度，非0为真。 如果`__len__`()也没有定义，那么所有实例都返回真

```
class A: pass
a = A()

print(bool(A))
print(bool(a))

class B:
    def __bool__(self):
        return False

print(bool(B))
print(bool(B()))
if B():
    print('Real B instance')

class C:
    def __len__(self):
        return 0

print(bool(C))
print(bool(C()))
if C():
    print('Real C instance')
```

运算符重载

operator模块提供以下的特殊方法，可以将类的实例使用下面的操作符来操作

运算符	特殊方法	含义
<, <=, ==, >, >=, !=	`__lt__`, `__le__`, `__eq__`, `__gt__`, `__ge__`, `__ne__`	比较运算符
+, -, *, /, %, //, **, divmod	`__add__`, `__sub__`, `__mul__`, `__truediv__`, `__mod__`, `__floordiv__`, `__pow__`, `__divmod__`	算数运算符，移位、位运算也有对应的方法
+=, -=, *=, /=, %=, //=, **=	`__iadd__`, `__isub__`, `__imul__`, `__itruediv__`, `__imod__`, `__ifloordiv__`, `__ipow__`	

实现自定义类的实例的大小比较（非常重要，排序时使用）

```
class A:
    pass

print(A() == A()) # 可以吗？
print(A() > A()) # 可以吗？
```

```
class A:
    def __init__(self, name, age=18):
        self.name = name
        self.age = age

    def __eq__(self, other):
        return self.name == other.name and self.age == other.age

    def __gt__(self, other):
        return self.age > other.age

    def __ge__(self, other):
        return self.age >= other.age
tom = A('tom')
jerry = A('jerry', 16)
print(tom == jerry, tom != jerry)
print(tom > jerry, tom < jerry)
print(tom >= jerry, tom <= jerry)
```

`__eq__`等于可以推断不等于
`__gt__`大于可以推断小于
`__ge__`大于等于可以推断小于等于
也就是用3个方法，就可以把所有比较解决了

实现两个学生的成绩差

```python
class A:
    def __init__(self, name, score):
        self.name = name
        self.score = score

tom = A('tom', 80)
jerry = A('jerry', 85)
print(tom.score - jerry.score)
```

```python
class A:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def __sub__(self, other):
        return self.score - other.score

tom = A('tom', 80)
jerry = A('jerry', 85)
print(tom.score - jerry.score)
print(tom - jerry)
print('~~~~~~~~~~~~~~~~~~~~~~~~~~')

jerry -= tom # 调用什么
print(tom)
print(jerry) # 显示什么
```

```python
class A:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def __sub__(self, other):
        return self.score - other.score

    def __isub__(self, other):
        #return A(self.name, self.score - other.score)
        self.score -= other.score
        return self

    def __repr__(self):
        return "<A name={}, score={}>".format(self.name, self.score)

tom = A('tom', 80)
jerry = A('jerry', 85)
print(tom.score - jerry.score)
print(tom - jerry)
print('~~~~~~~~~~~~~~~~~~~~~~~~~~')

jerry -= tom # 调用什么
print(tom)
print(jerry)
```

思考：list的+和+=的区别。tuple呢?