

魔术方法 ***

实例化

方法	意义
<code>__new__</code>	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 <code>__init__</code> 该方法永远都是静态方法

```
1 class A:
2     def __new__(cls, *args, **kwargs):
3         print(cls)
4         print(args)
5         print(kwargs)
6         #return super().__new__(cls)
7         #return 1
8         return None
9
10    def __init__(self, name):
11        self.name = name
12
13    a = A()
14    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
<code>__str__</code>	<code>str()</code> 函数、 <code>format()</code> 函数、 <code>print()</code> 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 <code>__repr__</code> 方法返回字符串表达，如果 <code>__repr__</code> 没有定义，就直接返回对象的内存地址信息
<code>__repr__</code>	内建函数 <code>repr()</code> 对一个对象获取 字符串 表达。 调用 <code>__repr__</code> 方法返回字符串表达，如果 <code>__repr__</code> 也没有定义，就直接返回object的定义就是显示内存地址信息
<code>__bytes__</code>	<code>bytes()</code> 函数调用，返回一个对象的bytes表达，即返回bytes对象

```
1 class A:
2     def __init__(self, name, age=18):
3         self.name = name
4         self.age = age
5
6     def __repr__(self):
7         return 'repr: {},{}'.format(self.name, self.age)
```

```

8
9     def __str__(self):
10         return 'str: {}'.format(self.name, self.age)
11
12     def __bytes__(self):
13         #return "{} is {}".format(self.name, self.age).encode()
14         import json
15         return json.dumps(self.__dict__).encode()
16
17
18 print(A('tom')) # print函数使用__str__
19 print('{}'.format(A('tom'))
20 print([A('tom')]) # []使用__str__, 但其内部使用__repr__
21 print([str(A('tom'))]) # []使用__str__, 其中的元素使用str()函数也调用__str__
22 print(bytes(A('tom'))

```

bool

方法	意义
<code>__bool__</code>	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

```

1 class A: pass
2 a = A()
3
4 print(bool(A))
5 print(bool(a))
6
7 class B:
8     def __bool__(self):
9         return False
10
11 print(bool(B))
12 print(bool(B()))
13 if B():
14     print('Real B instance')
15
16 class C:
17     def __len__(self):
18         return 0
19
20 print(bool(C))
21 print(bool(C()))
22 if C():
23     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
<, <=, ==, >, >=, !=	<code>__lt__</code> , <code>__le__</code> , <code>__eq__</code> , <code>__gt__</code> , <code>__ge__</code> , <code>__ne__</code>	比较运算符
+, -, *, /, %, //, **, divmod	<code>__add__</code> , <code>__sub__</code> , <code>__mul__</code> , <code>__truediv__</code> , <code>__mod__</code> , <code>__floordiv__</code> , <code>__pow__</code> , <code>__divmod__</code>	算数运算符，移位、位运算也有对应的方法
+=, -=, *=, /=, %=, //=, **=	<code>__iadd__</code> , <code>__isub__</code> , <code>__imul__</code> , <code>__itruediv__</code> , <code>__imod__</code> , <code>__ifloordiv__</code> , <code>__ipow__</code>	

实现自定义类的实例的大小比较（非常重要，排序时使用）

```
1 class A:
2     pass
3
4 print(A() == A()) # 可以吗?
5 print(A() > A()) # 可以吗?
```

```
1 class A:
2     def __init__(self, name, age=18):
3         self.name = name
4         self.age = age
5
6     def __eq__(self, other):
7         return self.name == other.name and self.age == other.age
8
9     def __gt__(self, other):
10        return self.age > other.age
11
12    def __ge__(self, other):
13        return self.age >= other.age
14    tom = A('tom')
15    jerry = A('jerry', 16)
16    print(tom == jerry, tom != jerry)
17    print(tom > jerry, tom < jerry)
18    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

```

1 class A:
2     def __init__(self, name, score):
3         self.name = name
4         self.score = score
5
6 tom = A('tom', 80)
7 jerry = A('jerry', 85)
8 print(tom.score - jerry.score)

```

```

1 class A:
2     def __init__(self, name, score):
3         self.name = name
4         self.score = score
5
6     def __sub__(self, other):
7         return self.score - other.score
8
9 tom = A('tom', 80)
10 jerry = A('jerry', 85)
11 print(tom.score - jerry.score)
12 print(tom - jerry)
13 print('~~~~~')
14
15 jerry -= tom # 调用什么
16 print(tom)
17 print(jerry) # 显示什么

```

```

1 class A:
2     def __init__(self, name, score):
3         self.name = name
4         self.score = score
5
6     def __sub__(self, other):
7         return self.score - other.score
8
9     def __isub__(self, other):
10        #return A(self.name, self.score - other.score)
11        self.score -= other.score
12        return self
13
14    def __repr__(self):
15        return "<A name={}, score={}>".format(self.name, self.score)
16
17 tom = A('tom', 80)
18 jerry = A('jerry', 85)
19 print(tom.score - jerry.score)
20 print(tom - jerry)
21 print('~~~~~')
22
23 jerry -= tom # 调用什么
24 print(tom)
25 print(jerry)

```

思考：list的+和+=的区别。tuple呢？

