1.14 排序不支持原生比较的对象¶

问题¶

你想排序类型相同的对象，但是他们不支持原生的比较操作。

解决方案¶

内置的 `sorted()` 函数有一个关键字参数 `key` ，可以传入一个 `callable` 对象给它， 这个 `callable` 对象对每个传入的对象返回一个值，这个值会被 `sorted` 用来排序这些对象。 比如，如果你在应用程序里面有一个 `User` 实例序列，并且你希望通过他们的 `user_id` 属性进行排序， 你可以提供一个以 `User` 实例作为输入并输出对应 `user_id` 值的 `callable` 对象。比如：

```
class User:
    def __init__(self, user_id):
        self.user_id = user_id

    def __repr__(self):
        return 'User({})'.format(self.user_id)


def sort_notcompare():
    users = [User(23), User(3), User(99)]
    print(users)
    print(sorted(users, key=lambda u: u.user_id))
```

另外一种方式是使用 `operator.attrgetter()` 来代替 lambda 函数：

```
>>> from operator import attrgetter
>>> sorted(users, key=attrgetter('user_id'))
[User(3), User(23), User(99)]
>>>
```

讨论¶

选择使用 lambda 函数或者是 `attrgetter()` 可能取决于个人喜好。 但是， `attrgetter()` 函数通常会运行的快点，并且还能同时允许多个字段进行比较。 这个跟 `operator.itemgetter()` 函数作用于字典类型很类似（参考1.13小节）。 例如，如果 `User` 实例还有一个 `first_name` 和 `last_name` 属性，那么可以向下面这样排序：

```
by_name = sorted(users, key=attrgetter('last_name', 'first_name'))
```

同样需要注意的是，这一小节用到的技术同样适用于像 `min()` 和 `max()` 之类的函数。比如：

```
>>> min(users, key=attrgetter('user_id'))
User(3)
>>> max(users, key=attrgetter('user_id'))
User(99)
>>>
```