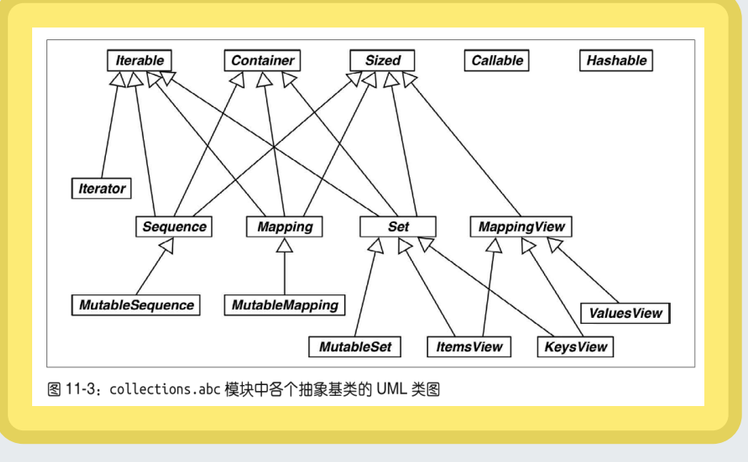
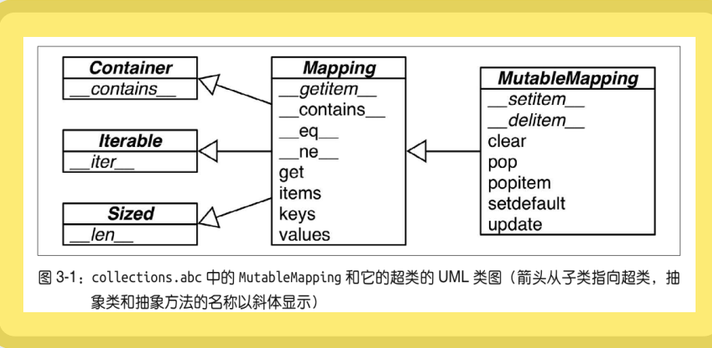
接口：从协议到抽象基类

- Python文化中的接口和协议
  - Python 语言没有interface 关键字，而且除了抽象基类，每个类都有接口：类实现或继承的公开属性（方法或数据属性），包括特殊方法，如 __getitem__ 或 __add__。
  - 按照定义，受保护的属性和私有属性不在接口中：即便“受保护的”属性也只是采用命名约定实现的（单个前导下划线）
  - 不要觉得把公开数据属性放入对象的接口中不妥，因为如果需要，总能实现读值方法和设值方法，把数据属性变成特性，使用 obj.attr 句法的客户代码不会受到影响。
  - 关于接口，这里有个实用的补充定义：对象公开方法的子集，让对象在系统中扮演特定的角色。
    - 接口是实现特定角色的方法集合，这样理解正是 Smalltalk 程序员所说的协议
  - 协议是接口，但不是正式的（只由文档和约定定义），因此协议不能像正式接口那样施加限制
    - 一个类可能只实现部分接口，这是允许的。
- Python喜欢序列
  - Python 数据模型的哲学是尽量支持基本协议。对序列来说，即便是最简单的实现，Python 也会力求做到最好。
  - 鉴于序列协议的重要性，如果没有 __iter__ 和 __contains__ 方法，Python 会调用__getitem__ 方法，设法让迭代和 in 运算符可用。
- 使用猴子补丁在运行时实现协议
  - 猴子补丁：在运行时修改类或模块，而不改动源码。
    - 猴子补丁很强大，但是打补丁的代码与要打补丁的程序耦合十分紧密，而且往往要处理隐藏和没有文档的部分。
  - 协议是动态的：random.shuffle 函数不关心参数的类型，只要那个对象实现了部分可变序列协议即可。
    - 所需的方法后来提供也行（猴子补丁）。
- Alex Martelli的水禽
  - 如果实现的类体现了 numbers、collections.abc 或其他框架中抽象基类的概念，要么继承相应的抽象基类（必要时），要么把类注册到相应的抽象基类中。
  - 开始开发程序时，不要使用提供注册功能的库或框架，要自己动手注册
  - 如果必须检查参数的类型（这是最常见的），例如检查是不是“序列”，那就这样做：isinstance(the_arg, collections.abc.Sequence)
  - 继承抽象基类很简单，只需要实现所需的方法，这样也能明确表明开发者的意图。
    - 这一意图还能通过注册虚拟子类来实现。
  - 使用 isinstance 和 issubclass 测试抽象基类更为人接受。
    - 毕竟，如果某个组件没有继承抽象基类，事后还可以注册，让显式类型检查通过。
  - 然而，即便是抽象基类，也不能滥用 isinstance 检查，用得多了可能导致代码异味，即表明面向对象设计得不好。
  - 如果必须强制执行 API 契约，通常可以使用 isinstance 检查抽象基类。
  - 抽象基类是用于封装框架引入的一般性概念和抽象的，例如“一个序列”和“一个确切的数”。
- 定义抽象基类的子类
  - 继承 MutableSequence 的类必须实现 __delitem__ 方法，这是 MutableSequence 类的一个抽象方法。
  - 要想实现子类，我们可以覆盖从抽象基类中继承的方法，以更高效的方式重新实现。
    - 例如，__contains__ 方法会全面扫描序列，可是，如果你定义的序列按顺序保存元素，那就可以重新定义 __contains__ 方法，使用 bisect 函数做二分查找（参见 2.8 节），从而提升搜索速度。
- 标准库中的抽象基类
  - 大多数抽象基类在 collections.abc 模块中定义，不过其他地方也有。例如，numbers 和 io 包中有一些抽象基类。
  - collections.abc模块中的抽象基类
    - 
    - Iterable、Container 和 Sized
      - 各个集合应该继承这三个抽象基类，或者至少实现兼容的协议。
      - Iterable 通过 __iter__ 方法支持迭代，Container 通过 __contains__ 方法支持 in 运算符，Sized 通过 __len__ 方法支持 len() 函数。
    - Sequence、Mapping 和 Set
      - 这三个是主要的不可变集合类型，而且各自都有可变的子类。
      - 
    - MappingView
      - 在 Python 3 中，映射方法 .items()、.keys() 和 .values() 返回的对象分别是 ItemsView、KeysView 和 ValuesView 的实例。
      - 前两个类还从 Set 类继承了丰富的接口
    - Callable 和 Hashable
      - 这两个抽象基类与集合没有太大的关系，只不过因为 collections.abc 是标准库中定义抽象基类的第一个模块，而它们又太重要了，因此才把它们放到 collections.abc 模块中。
      - 这两个抽象基类的主要作用是为内置函数 isinstance 提供支持，以一种安全的方式判断对象能不能调用或散列。
    - Iterator
      - 注意它是 Iterable 的子类。
  - 抽象基类的数字塔
    - 线性层次的数字塔，Number 是位于最顶端的超类，最低端是 Integral 类。
      - Number
      - Complex
      - Real
      - Rational
      - Integral
    - 如果想检查一个数是不是整数，可以使用 isinstance(x, numbers.Integral)，这样代码就能接受 int、bool（int 的子类），或者外部库使用 numbers 抽象基类注册的其他类型。
      - 为了满足检查的需要，你或者你的 API 的用户始终可以把兼容的类型注册为 numbers.Integral 的虚拟子类。
    - 如果一个值可能是浮点数类型，可以使用 isinstance(x, numbers.Real) 检查。这样代码就能接受 bool、int、float、fractions.Fraction，或者外部库（如 NumPy，它做了相应的注册）提供的非复数类型。
- 定义并使用一个抽象基类
  - 实现定义抽象注意事项
    - 抽象方法使用 @abstractmethod 装饰器标记，而且定义体中通常只有文档字符串。
    - 抽象基类可以包含具体方法。
    - 抽象基类中的具体方法只能依赖抽象基类定义的接口（即只能使用抽象基类中的其他具体方法、抽象方法或特性）。
    - 抽象方法可以有实现代码。即便实现了，子类也必须覆盖抽象方法，但是在子类中可以使用 super() 函数调用抽象方法，为它添加功能，而不是从头开始实现。
  - 抽象基类句法详解
    - 声明抽象基类最简单的方式是继承 abc.ABC 或其他抽象基类。
      - abc.ABC 是 Python 3.4 新增的类
      - 旧版 Python
        - metaclass= 关键字参数是 Python 3 引入的。在 Python 2 中必须使用 __metaclass__ 类属性
        - 把值设为 abc.ABCMeta（不是 abc.ABC）
    - 在函数上堆叠装饰器的顺序通常很重要
      - 与其他方法描述符一起使用时，abstractmethod() 应该放在最里层，
      - 在 @abstractmethod 和 def 语句之间不能有其他装饰器。
  - 虚拟子类
    - 注册虚拟子类的方式是在抽象基类上调用 register 方法。
      - 注册的类会变成抽象基类的虚拟子类，而且 issubclass 和 isinstance 等函数都能识别，但是注册的类不会从抽象基类中继承任何方法或属性。
      - register 方法通常作为普通的函数调用（参见 11.9 节），不过也可以作为装饰器使用。
    - 虚拟子类不会继承注册的抽象基类，而且任何时候都不会检查它是否符合抽象基类的接口，即便在实例化时也不会检查。
    - 类的继承关系在一个特殊的类属性中指定——__mro__，即方法解析顺序（Method Resolution Order)。
      - 这个属性的作用很简单，按顺序列出类及其超类，Python 会按照这个顺序搜索方法。
- 子类的测试方法
  - __subclasses__() 这个方法返回类的直接子类列表，不含虚拟子类。
    - 即便源码中用不到想测试的模块，也要将其导入，因为要把那些类载入内存。
  - _abc_registry 只有抽象基类有这个数据属性，其值是一个 WeakSet 对象，即抽象类注册的虚拟子类的弱引用。
- Python使用register的方式
  - 如果是 Python 3.3 或之前的版本，不能把 .register 当作类装饰器使用，必须使用标准的调用句法。
  - 虽然现在可以把 register 当作装饰器使用了，但更常见的做法还是把它当作函数使用，用于注册其他地方定义的类。
- 鹅的行为有可能像鸭子
  - 即便不注册，抽象基类也能把一个类识别为虚拟子类。
  - 在你我自己编写的抽象基类中实现 __subclasshook__ 方法，可靠性很低。
    - 当然，自己实现的 __subclasshook__ 方法还可以检查方法签名和其他特性
- 白鹅类型
  - 白鹅类型指，只要 cls 是抽象基类，即 cls 的元类是 abc.ABCMeta，就可以使用 isinstance(obj, cls)。
  - 借助“白鹅类型”，可以使用抽象基类明确声明接口，而且类可以子类化抽象基类或使用抽象基类注册（无需在继承关系中确立静态的强链接），宣称它实现了某个接口。
  - 白鹅类型的一个基本特性（也是值得用水禽来命名的原因）：即便不继承，也有办法把一个类注册为抽象基类的虚拟子类。
  - __subclasshook__ 在白鹅类型中添加了一些鸭子类型的踪迹。