- 特殊方法
  - Python 解释器碰到特殊的句法时，会使用特殊方法去激活一些基本的对象操作，这些特殊方法的名字以两个下划线开头，以两个下划线结尾（例如 __getitem__）。
  - 迭代通常是隐式的，譬如说一个集合类型没有实现 __contains__ 方法，那么 in 运算符就会按顺序做一次迭代搜索。
  - 通过数据模型和一些合成来实现这些功能
  - 如何使用特殊方法
    - 特殊方法的存在是为了被 Python 解释器调用的，你自己并不需要调用它们。
    - 然而如果是 Python 内置的类型，比如列表（list）、字符串（str）、字节序列（bytearray）等，那么 CPython 会抄个近路，__len__ 实际上会直接返回 PyVarObject 里的 ob_size 属性。
    - 模拟数值类型
      - 利用特殊方法，可以让自定义对象通过加号“+”（或是别的运算符）进行运算。
    - 字符串表示形式
      - Python 有一个内置的函数叫 repr，它能把一个对象用字符串的形式表达出来以便辨认，这就是“字符串表示形式”。
      - __repr__
        - repr 就是通过 __repr__ 这个特殊方法来得到一个对象的字符串表示形式的。
        - 在 __repr__ 的实现中，我们用到了 %r 来获取对象各个属性的标准字符串表示形式
        - __repr__ 所返回的字符串应该准确、无歧义，并且尽可能表达出如何用代码创建出这个被打印的对象。
        - __repr__ 和 __str__ 的区别在于，后者是在 str() 函数被使用，或是在用 print 函数打印一个对象的时候才被调用的，并且它返回的字符串对终端用户更友好。
    - 算术运算符
      - __add__ 和 __mul__
      - __rmul__
        - 乘法的交换律
    - 自定义的布尔值
      - 任何对象都可以用于需要布尔值的上下文中（比如if 或 while 语句，或者 and、or 和 not 运算符）
      - 默认情况下，我们自己定义的类的实例总被认为是真的，除非这个类对 __bool__ 或者 __ len__ 函数有自己的实现。
      - bool(x) 的背后是调用 x.__bool__() 的结果；如果不存在 __ bool__ 方法，那么 bool(x) 会尝试调用 x.__len__()。
  - 特殊方法一览
    - 表1-1. 跟运算符无关的特殊方法

类别	方法名
字符串/字节序列表示形式	__repr__、__str__、__format__、__bytes__
数值转换	__abs__、__bool__、__complex__、__int__、__float__、__hash__、__index__
集合模拟	__len__、__getitem__、__setitem__、__delitem__、__contains__
迭代协议	__iter__、__reversed__、__next__
可调用模拟	__call__
上下文管理	__enter__、__exit__
实例创建和销毁	__new__、__init__、__del__
属性管理	__getattr__、__getattribute__、__setattr__、__delattr__、__dir__
属性描述符	__get__、__set__、__delete__
跟类相关的服务	__prepare__、__instancecheck__、__subclasscheck__

    - 表1-2. 跟运算符相关的特殊方法

类别	方法名和对应的运算符
一元运算符	__neg__ -、__pos__ +、__abs__ abs()
众多比较运算符	__lt__ <、__le__ <=、__eq__ ==、__ne__ !=、__gt__ >、__ge__ >=
算术运算符	__add__ +、__sub__ -、__mul__ *、__truediv__ /、__floordiv__ //、__mod__ %、__divmod__ divmod()、__pow__ ** 或 pow()、__round__ round()
反向算术运算符	__radd__、__rsub__、__rmul__、__rtruediv__、__rfloordiv__、__rmod__、__rdivmod__、__rpow__
增量赋值算术运算符	__iadd__、__isub__、__imul__、__itruediv__、__ifloordiv__、__imod__、__ipow__
位运算符	__invert__ ~、__lshift__ <<、__rshift__ >>、__and__ &、__or__ |、__xor__ ^
反向位运算符	__rlshift__、__rrshift__、__rand__、__rxor__、__ror__
增量赋值位运算符	__ilshift__、__irshift__、__iand__、__ixor__、__ior__

  - 为什么len不是普通方法
    - 是为了让 Python 自带的数据结构可以走后门
      - 如果 x 是一个内置类型的实例，那么 len(x) 的速度会非常快。背后的原因是 CPython 会直接从一个 C 结构体里读取对象的长度，完全不会调用任何方法。
    - 可以把 len 用于自定义数据类型
    - 这种处理方式在保持内置类型的效率和保证语言的一致性之间找到了一个平衡点
  - 本章小结
    - __repr__ 和__str__
      - Python 对象的一个基本要求就是它得有合理的字符串表示形式
      - 前者方便我们调试和记录日志，后者则是给终端用户看的。
    - 对序列数据类型的模拟是特殊方法用得最多的地方
    - Python 通过运算符重载这一模式提供了丰富的数值类型，除了内置的那些之外，还有 decimal.Decimal 和 fractions.Fraction。
      - 这些数据类型都支持中缀算术运算符