- 3-从函数到方法（2）
  - 方法
    - 1-基本的方法
      - __init__(self[, ...]) 构造器，当一个实例被创建的时候调用的初始化方法
      - __new__(cls[, ...]) 在一个对象实例化的时候所调用的第一个方法，在调用__init__初始化前，先调用__new__。
__new__至少要有一个参数cls，代表要实例化的类，此参数在实例化时由 Python 解释器自动提供，后面的参数直接传递给__init__。
__new__对当前类进行了实例化，并将实例返回，传给__init__的self。但是，执行了__new__，并不一定会进入__init__，只有__new__返回了，当前类cls的实例，当前类的__init__才会进入。
      - 若__new__没有正确返回当前类cls的实例，那__init__是不会被调用的，即使是父类的实例也不行，将没有__init__被调用。
      - __new__方法主要是当你继承一些不可变的 class 时（比如int, str, tuple）， 提供给你一个自定义这些类的实例化过程的途径。
      - __del__(self) 析构器，当一个对象将要被系统回收之时调用的方法。
      - __str__(self):
        - ·当你打印一个对象的时候，触发__str__
        - ·当你使用%s格式化的时候，触发__str__
        - ·str强转数据类型的时候，触发__str__
      - __repr__(self):
        - ·repr是str的备胎
        - ·有__str__的时候执行__str__,没有实现__str__的时候，执行__repr__
        - ·repr(obj)内置函数对应的结果是__repr__的返回值
        - ·当你使用%r格式化的时候 触发__repr__
      - 类型工厂函数，指的是“不通过类而是通过函数来创建对象”
    - 2-算数运算符
      - __add__(self, other)定义加法的行为：+
__sub__(self, other)定义减法的行为：-
      - __mul__(self, other)定义乘法的行为：*
__truediv__(self, other)定义真除法的行为：/
__floordiv__(self, other)定义整数除法的行为：//
__mod__(self, other) 定义取模算法的行为：%
__divmod__(self, other)定义当被 divmod() 调用时的行为
divmod(a, b)把除数和余数运算结果结合起来，返回一个包含商和余数的元组(a // b, a % b)。
      - __pow__(self, other[, module])定义当被 power() 调用或 ** 运算时的行为
__lshift__(self, other)定义按位左移位的行为：<<
__rshift__(self, other)定义按位右移位的行为：>>
      __and__(self, other)定义按位与操作的行为：&
__xor__(self, other)定义按位异或操作的行为：^
__or__(self, other)定义按位或操作的行为：|
    - 3-反算术运算符
      - 反运算魔方方法，与算术运算符保持一一对应，不同之处就是反运算的魔法方法多了一个“r”。当文件左操作不支持相应的操作时被调用。
      - __radd__(self, other)定义加法的行为：+
__rsub__(self, other)定义减法的行为：-
__rmul__(self, other)定义乘法的行为：*
__rtruediv__(self, other)定义真除法的行为：/
__rfloordiv__(self, other)定义整数除法的行为：//
__rmod__(self, other) 定义取模算法的行为：%
__rdivmod__(self, other)定义当被 divmod() 调用时的行为
__rpow__(self, other[, module])定义当被 power() 调用或 ** 运算时的行为
__rlshift__(self, other)定义按位左移位的行为：<<
__rrshift__(self, other)定义按位右移位的行为：>>
      __rand__(self, other)定义按位与操作的行为：&
__rxor__(self, other)定义按位异或操作的行为：^
__ror__(self, other)定义按位或操作的行为：|
    - 4-增量赋值运算符
      - __iadd__(self, other)定义赋值加法的行为：+=
__isub__(self, other)定义赋值减法的行为：-=
__imul__(self, other)定义赋值乘法的行为：*=
__itruediv__(self, other)定义赋值真除法的行为：/=
__ifloordiv__(self, other)定义赋值整数除法的行为：//=
__imod__(self, other)定义赋值取模算法的行为：%=
__ipow__(self, other[, modulo])定义赋值幂运算的行为：**=
__ilshift__(self, other)定义赋值按位左移位的行为：<<=
__irshift__(self, other)定义赋值按位右移位的行为：>>=
__iand__(self, other)定义赋值按位与操作的行为：&=
__ixor__(self, other)定义赋值按位异或操作的行为：^=
__ior__(self, other)定义赋值按位或操作的行为：|=
    - 5-一元运算符
      - __neg__(self)定义正号的行为：+x
__pos__(self)定义负号的行为：-x
__abs__(self)定义当被abs()调用时的行为
__invert__(self)定义按位求反的行为：~x
    - 6-属性访问
      - __getattr__(self, name): 定义当用户试图获取一个不存在的属性时的行为。
__getattribute__(self, name)：定义当该类的属性被访问时的行为（先调用该方法，查看是否存在该属性，若不存在，接着去调用__getattr__）。
__setattr__(self, name, value)：定义当一个属性被设置时的行为。
__delattr__(self, name)：定义当一个属性被删除时的行为。
    - 7-描述符
      - 描述符就是将某种特殊类型的类的实例指派给另一个类的属性。
      - __get__(self, instance, owner)用于访问属性，它返回属性的值。
__set__(self, instance, value)将在属性分配操作中调用，不返回任何内容。
__del__(self, instance)控制删除操作，不返回任何内容。
      - 协议（Protocols）与其它编程语言中的接口很相似，它规定你哪些方法必须要定义。然而，在 Python 中的协议就显得不那么正式。事实上，在 Python 中，协议更像是一种指南。
    - 8-协议
      - 容器类型的协议
        - ·如果说你希望定制的容器是不可变的话，你只需要定义__len__()和__getitem__()方法。
        - ·如果你希望定制的容器是可变的话，除了__len__()和__getitem__()方法，你还需要定义__setitem__()和__delitem__()两个方法。
    - 9-迭代器
      - 迭代是 Python 最强大的功能之一，是访问集合元素的一种方式。
迭代器是一个可以记住遍历的位置的对象。
迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问完结束。
迭代器只能往前不会后退。
字符串，列表或元组对象都可用于创建迭代器：
      - 迭代器有两个基本的方法：iter() 和 next()。
iter(object) 函数用来生成迭代器。
next(iterator[, default]) 返回迭代器的下一个项目。
iterator -- 可迭代对象
default -- 可选，用于设置在没有下一个元素时返回该默认值，如果不设置，又没有下一个元素则会触发 StopIteration 异常。
      - 把一个类作为一个迭代器使用需要在类中实现两个方法 __iter__() 与 __next__() 。
        - __iter__(self)定义当迭代容器中的元素的行为，返回一个特殊的迭代器对象， 这个迭代器对象实现了 __next__() 方法并通过 StopIteration 异常标识迭代的完成。
__next__() 返回下一个迭代器对象。
StopIteration 异常用于标识迭代的完成，防止出现无限循环的情况，在 __next__() 方法中我们可以设置在完成指定循环次数后触发 StopIteration 异常来结束迭代。
    - 10-生成器
      - ·在 Python 中，使用了 yield 的函数被称为生成器（generator）。
      - ·跟普通函数不同的是，生成器是一个返回迭代器的函数，只能用于迭代操作，更简单点理解生成器就是一个迭代器。
      - ·在调用生成器运行的过程中，每次遇到 yield 时函数会暂停并保存当前所有的运行信息，返回 yield 的值, 并在下一次执行 next() 方法时从当前位置继续运行。
      - ·调用一个生成器函数，返回的是一个迭代器对象。