- 协程
  - 从根本上把 yield 视作控制流程的方式，这样就好理解协程了。
  - 用作协程的生成器的基本行为
    - 如果协程只需从客户那里接收数据，那么产出的值是 None——这个值是隐式指定的，因为 yield 关键字右边没有表达式。
    - 协程可以身处四个状态中的一个。当前状态可以使用 inspect.getgeneratorstate(...) 函数确定，该函数会返回下述字符串中的一个。
      - 'GEN_CREATED' 等待开始执行。
      - 'GEN_RUNNING' 解释器正在执行。
      - 'GEN_SUSPENDED' 在 yield 表达式处暂停。
      - 'GEN_CLOSED' 执行结束。
    - 因为 send 方法的参数会成为暂停的 yield 表达式的值，所以，仅当协程处于暂停状态时才能调用 send 方法
    - 协程在 yield 关键字所在的位置暂停执行
      - 前面说过，在赋值语句中，= 右边的代码在赋值之前执行。
      - 因此，对于 b = yield a 这行代码来说，等到客户端代码再激活协程时才会设定 b 的值。
  - 预激协程的装饰器
    - 使用协程之前必须预激
      - next(coroutine)
      - 为了避免忘记，可以在协程上使用一个特殊的装饰器。
    - 示例
    - 很多框架都提供了处理协程的特殊装饰器，不过不是所有装饰器都用于预激协程，有些会提供其他服务，例如与入事件循环。
      - 异步网络库 Tornado 提供了 tornado.gen 装饰器
    - 使用 yield from 句法（参见 16.7 节）调用协程时，会自动预激，因此与示例 16-5 中的 @coroutine 等装饰器不兼容。
      - Python 3.4 标准库里的 asyncio.coroutine 装饰器（第 18 章介绍）不会预激协程，因此能兼容 yield from 句法。
  - 终止协程和异常处理
    - 协程中未处理的异常会向上冒泡，传给 next 函数或 send 方法的调用方（即触发协程的对象）。
    - 终止协程的一种方式：发送某个哨符值，让协程退出。
      - 内置的 None 和Ellipsis 等常量经常用作哨符值。
        - Ellipsis 的优点是，数据流中不太常有这个值。
      - 有人把 StopIteration 类（类本身，而不是实例，也不抛出）作为哨符值
    - 从 Python 2.5 开始，客户代码可以在生成器对象上调用两个方法，显式地把异常发给协程。
      - generator.throw(exc_type[, exc_value[, traceback]])
        - 致使生成器在暂停的 yield 表达式处抛出指定的异常。
        - 如果生成器处理了抛出的异常，代码会向前执行到下一个 yield 表达式，而产出的值会成为调用 generator.throw 方法得到的返回值。
        - 如果生成器没有处理抛出的异常，异常会向上冒泡，传到调用方的上下文中。
      - generator.close()
        - 致使生成器在暂停的 yield 表达式处抛出 GeneratorExit 异常。
        - 如果生成器没有处理这个异常，或者抛出了 StopIteration 异常（通常是抛运行到结尾），调用方不会报错。
        - 如果收到 GeneratorExit 异常，生成器一定不能产出值，否则解释器会抛出RuntimeError 异常。
        - 生成器抛出的其他异常会向上冒泡，传给调用方。
    - try/catch 处理了异常，协程状态会变成'GEN_SUSPENDED'。
    - 如果传入协程的异常没有处理，协程会停止，即状态变成 'GEN_CLOSED'。
    - 如果不管协程如何结束都想做些清理工作，要把协程定义体中相关的代码放入 try/ finally 块中
  - 让协程返回值
    - 某些协程不会产出值，而是在最后返回一个值（通常是某种累计值）。
    - 在 Python 3.3 之前，如果生成器返回值，解释器会报句法错误。
  - 使用yield from
    - yield from 结构会在内部自动捕获 StopIteration 异常。
    - 在生成器 gen 中使用 yield from subgen() 时，subgen 会获得控制权，把产出的值传给 gen 的调用方，即调用方可以直接控制 subgen。与此同时，gen 会阻塞，等待 subgen 终止。
    - yield from x 表达式对 x 对象所做的第一件事是，调用 iter(x)，从中获取迭代器。
      - x 可以是任何可迭代的对象。
    - yield from 的主要功能是打开双向通道，把最外层的调用方与最内层的子生成器连接起来，这样二者可以直接发送和产出值，还可以直接传入异常，而不用在位于中间的协程中添加大量处理异常的样板代码。
    - 任何 yield from 链条都必须由客户驱动，在最外层委派生成器上调用 next(...) 函数或 .send(...) 方法。可以隐式调用，例如使用 for 循环。
    - 专门的术语
      - 委派生成器
        - 包含 yield from <iterable> 表达式的生成器函数。
      - 子生成器
        - 从 yield from 表达式中 <iterable> 部分获取的生成器。
      - 调用方
        - 指代调用委派生成器的客户端代码。
  - yield from的意义
    - 把迭代器当作生成器使用，相当于把子生成器的定义体内联在 yield from 表达式中。
    - yield from 的行为
      - 子生成器产出的值都直接传给委派生成器的调用方（即客户端代码）。
      - 使用 send() 方法发给委派生成器的值都直接传给子生成器。
        - 如果发送的值是 None，那么会调用子生成器的 __next__() 方法。
        - 如果发送的值不是 None，那么会调用子生成器的 send() 方法。
        - 如果调用的方法抛出 StopIteration 异常，那么委派生成器恢复运行。
          任何其他异常都会向上冒泡，传给委派生成器。
      - 生成器退出时，生成器（或子生成器）中的 return expr 表达式会触发 StopIteration(expr) 异常抛出。
      - yield from 表达式的值是子生成器终止时传给 StopIteration 异常的第一个参数。
    - yield from 结构的另外两个特性与异常和终止有关。
      - 传入委派生成器的异常，除了 GeneratorExit 之外都传给子生成器的 throw() 方法。
        - 如果调用 throw() 方法时抛出 StopIteration 异常，委派生成器恢复运行。StopIteration 之外的异常会向上冒泡，传给委派生成器。
      - 如果把 GeneratorExit 异常传入委派生成器，或者在委派生成器上调用 close() 方法，那么在子生成器上调用 close() 方法，如果它有的话。
        - 如果调用 close() 方法导致异常抛出，那么异常会向上冒泡，传给委派生成器；否则，委派生成器抛出 GeneratorExit 异常。
    - 子生成器可能只是纯粹的迭代器，不支持 .throw(...) 和 .close() 方法，因此 yield from 结构的逻辑必须处理这种情况。
      - 如果子生成器实现了这两个方法，而在子生成器内部，这两个方法都会触发异常抛出，这种情况也必须由 yield from 机制处理。
    - 调用方可能会无缘无故地让子生成器自己抛出异常，实现yield from 结构时也必须处理这种情况。
    - 为了优化，如果调用方调用 next(...) 函数或 .send(None) 方法，都要转交职责，在子生成器上调用 next(...) 函数；仅当调用方发送的值不是 None 时，才使用子生成器的 .send(...) 方法。