School of Economics and Management, Beihang University

现代程序设计技术

赵吉昌

本周内容

- 面向对象编程
  - 观察者模式
  - 生成器与迭代器

观察者模式

- 亦称
  - 发布（publish ）-订阅（Subscribe）模式
  - 模型-视图（View）模式
  - 源-收听者(Listener)模式
  - 从属者模式
- 要义
  - 一个目标对象管理所有依赖于它的观察者对象，并且在它本身的状态改变时主动发出通知
  - 观察者模式完美地将观察者和被观察的对象分离

观察者模式

- 优点
  - 观察者与被观察者之间抽象耦合
  - 可以触发多个符合单一职责的模块
  - 可以很方便地实现广播
- 场景
  - 消息交换，如消息队列；
  - 多级触发，如一个中断即会引发一连串反应
- 缺点
  - 效率不一定高

观察者模式

- Demo 1
  - ps1.py
- Demo 2
  - ps2.py

迭代

- 迭代（Iteration）
  - 通过for ... in等遍历数据结构如tuple, list, dict，字符串等
  - 判断一个对象是否可迭代
    - from collections import Iterable
    - isinstance([1,2,3], Iterable)
  - 同时迭代序号与元素
    - 内置的enumerate函数
    - for i, value in enumerate(['A', 'B', 'C']):
    - pass

生成器

- 生成器（generator）
  - 直接生成列表可能受到内存大小的限制，或者导致较高但不必要的时间成本
    - 需要“惰性求值”
    - 在循环的过程中不断返回后续元素
    - 避免一次创建完整的数据结构，从而节省大量的空间
  - 在Python中，这种一边循环一边计算元素的机制称为生成器

生成器

- 通过列表推导式构建生成器
  - 列表：`L=[x*x for x in range(10)]`
  - 生成器：`G=(x*x for x in range(10))`
  - 通过`next()`函数获得`generator`的下一个返回值
    - `next(G)`
  - 通过`for...in`进行遍历

生成器

- 通过定义函数构建生成器
  - 函数定义中须包含yield关键字
  - 在执行中遇到yield会中断，下次继续执行
    - 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
  - 获取返回值需要捕获StopIteration异常
  - 注意区分普通函数和generator函数
    - 普通函数调用直接返回结果
    - generator函数的“调用”实际返回一个generator对象
  - Demo: fib.py flatten.py

迭代器

- 迭代器（Iterator）
  - 可以被next()函数调用并不断返回下一个值的对象称为迭代器
    - 迭代器可以记住遍历位置
    - 迭代器只能往前不能后退(”消耗“数据）
  - 可以使用isinstance()判断一个对象是否是Iterator对象：
    - `isinstance((x for x in range(10)), Iterator)`

迭代器

- 迭代器（Iterator）
  - 生成器都是Iterator，但list、dict、str虽然是Iterable，却不是Iterator
  - 把list、dict、str等Iterable变成Iterator可以使用iter()函数
  - Iterator对象表示一个数据流
    - 被next()函数调用并不断返回下一个数据，直到没有数据时抛出StopIteration错误
    - 可将该数据流看做是长度未知的有序序列，只能不断通过next()函数实现按需计算下一个数据
      - 惰性计算
    - 可表示无限大数据流，例如全体自然数，而使用list等不可能存储全体自然数

迭代器

- 创建迭代器
  - iter()函数
  - 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__()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
  - __next__() 方法返回下一个元素并通过StopIteration 异常标识迭代的完成
  - Demo: numIter.py reverse.py

迭代器

- 迭代器相关工具（Demo:iter.py）
  - import itertools
  - compress(it,selector_it)
    - 并行处理两个可迭代对象，如果selector_it中的元素为真，则返回it中对应位置的元素
  - takewhile(predicate,it)
    - 不断使用当前元素作为参数调用predicate函数并测试返回结果，如果函数返回值为真，则生成当前元素，循环继续;否则立即中断当前循环
  - dropwhile(predicate,it)
    - 处理it,跳过predicate计算结果为True的元素后，不再进一步检查，输出剩下的元素

迭代器

- 迭代器相关工具
  - filterfalse 与filter相反
  - islice(it,stop) 作用类似于[:stop]
  - islice(it,start,stop,step=1)
  - accumulate(it)
    - 累计求和
  - starmap(func,it)
    - 把it中各个元素传给func
    - 类似于map(func,*element)

迭代器

- 迭代器相关工具
  - chain(it1,...,itN)
    - 返回一个生成器，依次产生it1...里的元素
  - chain.from_iterable(it)
    - it是一个由可迭代对象组成的可迭代对象，将之拆分并依次返回
  - product(it1,...,itN)
    - 从输入的各个可迭代对象中获取元素，合并成由N个元素组成的元组

迭代器

- 迭代器相关工具
  - 内置函数zip([iterable, ...])
    - 对应元素组合，元素个数与最短的列表一致
    - zip(*z)
  - zip_longest(fillvalue='fill’)
    - 元素个数与最长的列表一致
  - combinations(it,out_len)
    - 把it中out_len个元素的组合以元组的形式输出，不包同一元素的组合
  - combinations_with_replacement(it,out_len
    - 包含同一元素的组合

迭代器

- 迭代器工具
  - count(start=0,step=1)
    - 不断产生数字，可以是浮点
  - cycle(it)
    - 按顺序重复输出it中的各个元素
    - Demo: wpie.py
  - permutations(it,out_len=None)
    - 生成长度为out_len的it元素的所有排列
  - repeat(item,[times])
    - 重复不断生成times个指定元素

迭代器

- 迭代器工具
  - groupby(it,key=None)
    - 返回groupb (key,group)，其中key是分组标准，group是生成器，用于产生分组中的元素，须对输入的可迭代对象使用分组标准进行排序，否则输出会混乱
  - tee(it,n=2)
    - 产生n个迭代器，每个迭代器都和输入的可迭代对象it一致

本周作业

- 生成器和迭代器有两种常见的使用场景。
  - 一. 后项需要前项导出，且无法通过列表推导式生成。例如，时间序列中的“随机游走”便是一种满足上述条件的序列数据。其公式为$$X_t = \mu + X_{t-1} + w_t$$，其中μ为漂移量，w_{t}是满足某种条件的独立同分布的随机变量，这里假设其服从正态分布N(0, σ^2)。本题要求写出实现该功能的迭代器函数。具体要求如下：
    - 1. 实现random_walk生成器，输入参数μ, X_0, σ^2，N，函数将迭代返回N个随机游走生成的变量。
    - 2. 利用zip，实现拼合多个random_walk的生成器，以生成一组时间上对齐的多维随机游走序列。
  - 二. 需要迭代的内容数据量过大，无法一次性加载。例如，在图像相关的深度学习任务中，由于数据总量过大，一次性加载全部数据耗时过长、内存占用过大，因此一般会采用批量加载数据的方法。（注：实际应用中由于需要进行采样等操作，通常数据加载类的实现原理更接近字典，例如pytorch中的Dataset类。）现提供文件FaceImages.zip(http://vis-www.cs.umass.edu/fddb/originalPics.tar.gz，其中包含5000余张人脸图片。要求设计FaceDataset类，实现图片数据的加载。具体要求：
    - 1. 类接收图片路径列表
    - 2. 类支持将一张图片数据以ndarray的形式返回（可以利用PIL库实现）。
    - 3. 实现__iter__方法。
    - 4. 实现__next__方法，根据类内的图片路径列表，迭代地加载并以ndarray形式返回图片数据。
  - 请实现上述生成器和迭代器并进行测试。