School of Economics ang Management, Beihang University

现代程序设计技术

赵吉昌
jichang@buaa.edu.cn

关于大作业

- 中期进度
  - 部分同学仍未选题并开始
- 图片题目的讨论
  - 除了近似之外还有别的思路
- 注意使用面向对象编程
  - 体会各类设计模式的使用

本周内容

- 面向对象编程
  - 适配器模式
  - 生成器与迭代器

适配器模式

- 适配器模式(Adapter)
  - 将某个类的接口转换成客户端期望的另一个接口表示，目的是消除由于接口不匹配所造成的类的兼容性问题
  - 目标类(Target)
    - 定义客户所需的接口，可以是一个抽象类或接口，也可以是具体类
  - 适配器类(Adapter)
    - 转换器，通过调用另一个接口对Adaptee和Target进行适配
  - 适配者类(Adaptee)
    - 被适配类，包括了客户希望的业务方法

适配器模式

- Demo
  - ad.py
- 适用场景
  - 没有现成的代码
  - 利用既有组件可能成本更低
  - 版本升级与兼容性
    - 新版本：Adaptee，旧版本：Target，Adapter类：实现旧版本类与新版本类的兼容

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

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
  - 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

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，accumulate(it,func)把前两个元素传给func，然后依次把计算结果和下一个元素传给func
  - 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
  - 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(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