1.

哲学问题：如果用到了模式，说明抽象的不够深

python里所有集合都可以迭代。迭代器可以支持for循环、构建和扩展集合类型、逐行遍历文本文件、列表推导、字典推导、集合推导、元组拆包（能这么用？？）、用\*拆包实参。

3.

（小知识点）reprlib.repr用于生成大型数据结构的简略字符串表示形式，默认最多生成30个字符。【虽然目前对我没啥用，好像还挺好玩】

4.

iter函数会做的事情：

1. 检查对象是否实现了\_\_iter\_\_，如果有就调用这个方法。
2. 如果没\_\_iter\_\_但是有\_\_getitem\_\_，python会尝试**从索引0**创建迭代器获取元素。
3. 如果都不行，扔出TypeError异常。

5.

可迭代对象：使用iter内置函数可以获取迭代器的对象。

实现了\_\_iter\_\_/序列/实现\_\_getitem\_\_**且参数从0开始索引** 这样的是可迭代的。

6.

P334对比了for和while处理可迭代对象的不同。可以看见while方式处理时要显式的整出迭代器，然后捕捉StopIteration异常。【有空把图粘上】

迭代器的接口有两个：

\_\_next\_\_返回下一个可用元素，如果没有，则抛出异常。

和\_\_iter\_\_返回self。如for里就会触发\_\_iter\_\_。

8.

P335图，iterable是父。

9.

P337，迭代器用完要重构。重构方法是用iter和之前的iterable对象返回一个新的迭代器。用iter处理迭代器本身是没用，因为它只是return self。

10.

P337里面说的第二个方法里，为了实现\_\_iter\_\_，专门给它写了一个用来返回的迭代器。写的迭代器里，分别实现了\_\_next\_\_和\_\_iter\_\_，其中\_\_next\_\_完全按照行为描述了一个迭代器，当然这也要\_\_init\_\_函数的协助。总之这一版很多代码在处理迭代器的内部状态。

11.

有时候构建可迭代对象就足够了，换言之不能实现\_\_next\_\_，\_\_iter\_\_也不是只返回自己。

12.

P339，第三版是用生成器函数代替第二版中的自己写的迭代器类。写法见截图【】。

13.

P340，yield函数是生成器工厂。

生成器理解成一次只触发一个yield，产生一个值？执行完毕后，它会抛出StopIteration异常。

14.

P343，代码更简短了，大概是用了finditer的API。其实也可以用生成器表达式（如下文）。

15.

P344，一开始没理解为什么是AAA,BBB,其实把它想成一个普通的迭代器就好了。

另一个值得注意的地方就是列表推导的输出结果和生成器表达式的区别。列表推导会马上“执行”生成器来获取结果。而生成器推导时，生成器不会马上执行，当进入for之后，原生成器才会工作。

16.

生成器函数和生成器表达式是等效的，有时后者更便利。

P345是生成器表达式版的代码

17.

P347的代码我应该自己也能写出来。

18.

P348，itertools模块里提供了很多生成器函数。

【把图截过来！！】

【自己实践一下，看看有没有哪些是可替代的关系！！】

19.

P357，yield from是从迭代器的迭代器里产生值，看起来像语法糖，但后面还有别的用。

20.

归约函数：输入一个可迭代的对象，返回单个结果。

【贴图】

【并补上sorted】

21.

iter+哨符看起来很酷炫，相当于对于一个函数，进行“满足条件就停止”的限制。

【贴例程】

22.

协程和迭代无关。

.send()，可以让生成器前进到下一个yield，不管给send()的是什么参数，这个值都会成为对应yield产生的值。