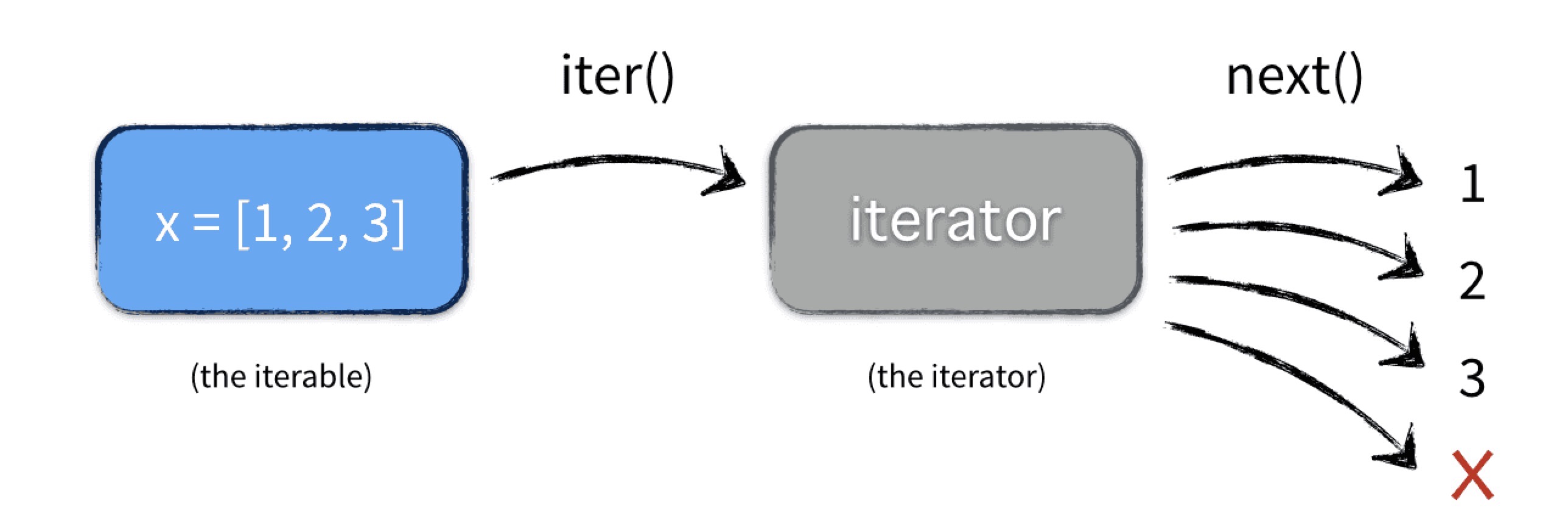
# 迭代器和生成器

## 容器

container 可以理解为把多个元素组织在一起的数据结构，container 中的元素可以逐个地迭代获取，可以用 in, not in 关键字判断元素是否包含在容器中。在 Python 中，常见的 container 对象有：

list, deque, .... set, frozensets, .... dict, defaultdict, OrderedDict, Counter, .... tuple, namedtuple, … str

## 可迭代对象（iterables）vs 迭代器（iterator）



迭代器

大部分的 container 都是可迭代对象，比如 list or set 都是可迭代对象，可以说只要是可以返回一个迭代器的都可以称作可迭代对象。下面看一个例子：

>>> x = [1, 2, 3]  
>>> y = iter(x)  
>>> next(y)  
1  
>>> next(y)  
2  
>>> type(x)  
<class 'list'>  
>>> type(y)  
<class 'list\_iterator'>

可见， x 是可迭代对象，这里也叫 container。y 则是迭代器，且实现了\_\_iter\_\_ 和 \_\_next\_\_方法。它们之间的关系是：

那什么是迭代器了？上面例子中有 2 个方法 iter and next。可见通过 iter 方法后就是迭代器。

它是一个带状态的对象，调用 next 方法的时候返回容器中的下一个值，可以说任何实现了iter和 next 方法的对象都是迭代器，iter返回迭代器自身，next 返回容器中的下一个值，如果容器中没有更多元素了，则抛异常。

迭代器就像一个懒加载的工厂，等到有人需要的时候才给它生成值返回，没调用的时候就处于休眠状态等待下一次调用。

## 生成器

迭代器和生成器

迭代器和生成器

生成器一定是迭代器，是一种特殊的迭代器，特殊在于它不需要再像上面的 iter() 和 next 方法了，只需要一个 yiled 关键字。下面来看一个例子：

用生成器实现斐波拉契

# content of test.py  
def fib(n):  
 prev, curr = 0, 1  
 while n > 0:  
 yield curr  
 prev, curr = curr, curr + prev  
 n -= 1

到终端执行 fib 函数

>>> from test import fib  
>>> y = fib(10)  
>>> next(y)  
1  
>>> type(y)  
<class 'generator'>  
>>> next(y)  
1  
>>> next(y)  
2

fib 就是一个普通的 Python 函数，它特殊的地方在于函数体中没有 return 关键字，函数的返回值是一个生成器对象(通过 yield 关键字)。当执行 f=fib() 返回的是一个生成器对象，此时函数体中的代码并不会执行，只有显示或隐示地调用 next 的时候才会真正执行里面的代码。

假设有千万个对象，需要顺序调取，如果一次性加载到内存，对内存是极大的压力，有生成器之后，可以需要的时候去生成一个，不需要的则也不会占用内存。

平常可能还会遇到一些生成器表达式，比如：

>>> a = (x\*x for x in range(10))  
>>> a  
<generator object <genexpr> at 0x102d79a20>  
>>> next(a)  
0  
>>> next(a)  
1  
>>> a.close()  
>>> next(a)  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
StopIteration

这些小技巧也是非常有用的。close 可以关闭生成器。生成器中还有一个 send 方法，其中 send(None) 与 next 是等价的。

>>> def double\_inputs():  
... while True:  
... x = yield  
... yield x \* 2  
...  
>>> generator = double\_inputs()  
>>> generator.send(10)  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
TypeError: can't send non-None value to a just-started generator  
>>> generator.send(None)  
>>> generator.send(10)  
20  
>>> next(generator)  
>>> generator.send(20)  
40

从上面的例子中可以看出，生成器可以接收参数，通过 send(value) 方法，且第一次不能直接 send(value)，需要 send(None) 或者 next() 执行之后。也就是说调用 send 传入非 None 值前，生成器必须处于挂起状态，否则将抛出异常。

## 区别

* 迭代器是一个更抽象的概念，任何对象，如果它有next方法（next python3，python2 是 \_\_next\_\_方法)和\_\_iter\_\_方法，则可以称作迭代器。
* 每个生成器都是一个迭代器，但是反过来不行。通常生成器是通过调用一个或多个yield表达式构成的函数生成的。同时满足迭代器的定义。
* 生成器能做到迭代器能做的所有事,而且因为自动创建了iter()和 next()方法,生成器显得特别简洁,而且生成器也是高效的。

## 参考资料

《全面剖析 Python 面试知识点》

https://gitbook.cn/books/5ca40fd11763103ff10b0e43/index.html