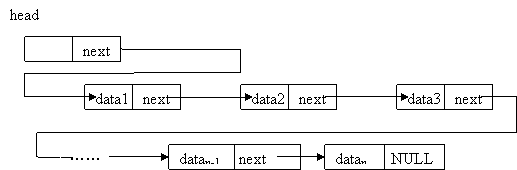
**链表**

1. 链表(linked-list)：链表就是线性表的链式存储方式。链表的内存是不连续的，前一个元素存储地址的下一个地址中存储的不一定是下一个元素。链表通过一个指向下一个元素地址的引用将链表中的元素串起来
2. 单向链表：

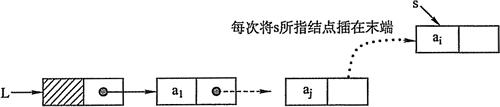
节点(node)：是链表中最基本的数据，每一个节点包含了数据块和指向下一个节点的指针。

头结点：单向链表有时候也分为有头结点和无头结点。有头结点的链表实现比较方便(每次插入新元素的时候，不需要每次判断第一个节点是否为空)，并且可以直接在头结点的数据块部分存储链表的长度，而不用每次都遍历整个链表。

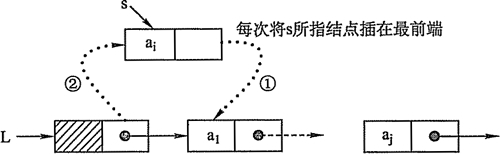


插入：在链表中插入一个新的元素有两种方式：后插和前插。后插就是每次在链表的末尾插入新元素，前插就是在链表的头插入新元素。

后插法比较符合平常的思维方式，并且保证插入数据的先后顺序。但是由于只保存了头结点，所以每次插入新元素必须重新遍历到链表末尾。为了解决这个问题，考虑增加一个尾指针，指向链表的最后一个节点。



由于前插法是在头部插入新元素，那么每次增加新元素可以直接通过头指针索引，但是得到的元素顺序与插入顺序相反。

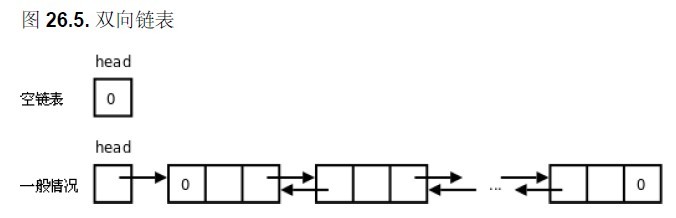


删除：由于单向链表只存储了头指针，所以删除单向链表中的元素时，需要找到目标节点的前驱节点

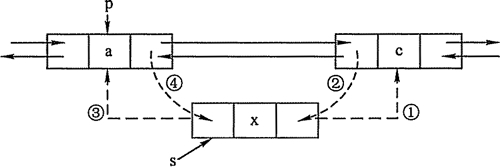
清空链表：由于链表里面的内存是手动分配的，当不再使用这些内存时需要手动删除。

1. 双向链表

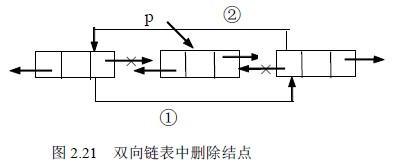
顾名思义，双向链表就是有两个方向的链表。同单向链表不同，在双向链表中每一个节点不仅存储指向下一个节点的指针，而且存储指向前一个节点的指针。通过这种方式，能够通过在O(1)时间内通过目的节点直接找到前驱节点，但是同时会增加大量的指针存储空间。



插入：在双向链表中插入新元素的操作跟在单向链表中插入新元素的操作类似

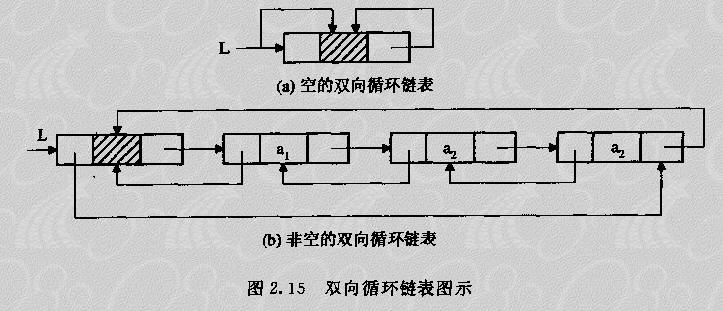


删除：由于双向链表中每个节点记录了它的前驱结点，所以不需要像单向链表中一样索引目的节点的前驱节点，而是可以通过目标节点直接获得。



1. 循环链表

循环链表与双向链表相似，不同的地方在于：在链表的尾部增加一个指向头结点的指针，头结点也增加一个指向尾节点的指针，以及第一个节点指向头节点的指针，从而更方便索引链表元素。



循环链表的插入和删除操作与双向链表的实现方式一样。

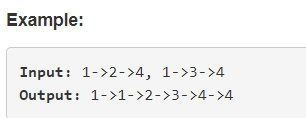
判断空列表：从上图(a)中可以明显观察到，一个空的双向循环链表中只有一个头节点，头节点的前驱和后驱都指向本身。

从图(b)中可以看到，不同于双向链表，循环链表中第一个节点和尾节点不在指向空指针，而是指向了头节点。

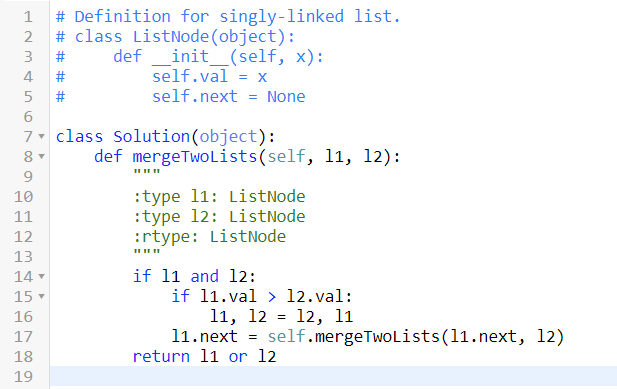
习题

1. 合并两个有序链表（题号21，Merge Two Sorted Lists）

题目：将连个已排序的链表按顺序合并成一个链表

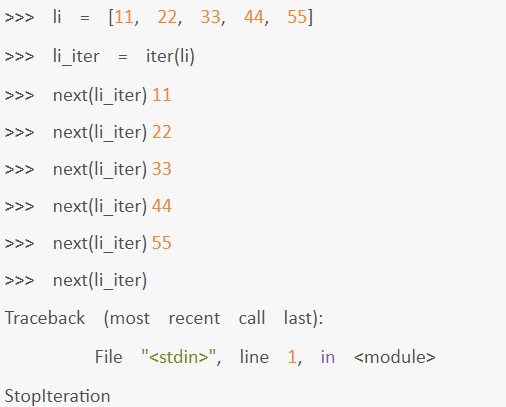


思路：python中并无指针，因此它需要模拟链表，比较常用的方式是先定义类，然后一个储存元素，一个储存地址。此题是将两个链表按元素大小顺序合并，让l1的第一个值与l2相比，若l1大，则交换l1和l2，若不大，则取l1的下一个值继续与l2的第一个值相比，这样就相当于在l1中遇到更小的后一半就替换了选来的后一半，然后再从新的后一半开始接着比，这样最后l1和l2中有一个就连接满了，一个为空，输出非空的即可



关于next:

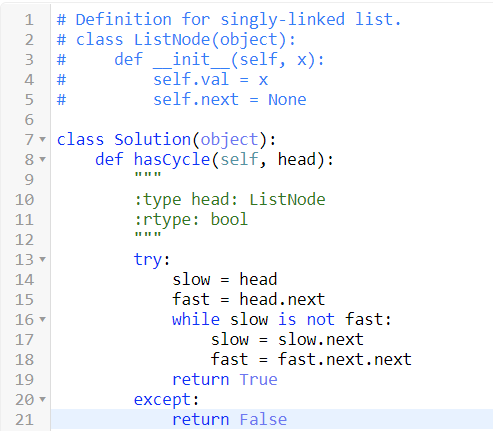
list、tuple等都是可迭代对象，我们可以通过iter()函数获取这些可迭代对象的迭代器。然后我们可以对获取到的迭代器不断使⽤next()函数来获取下⼀条数据。iter()函数实际上就是调⽤了可迭代对象的\_\_iter\_\_⽅法。



1. 循环链表（题号141，Linked List Cycle）

题目：判断一个链表是否为循环的，要求不占用额外空间

思路：双指针思路：一个指针慢，一个一个走，另一个指针快，两个两个走，若是循环链表，这两个指针一定会相遇，就像赛跑的两个人一定会相遇，若不是循环链表，快指针一定会先指空，一定会停止



Try在这里起到防止fast和slow为空的作用