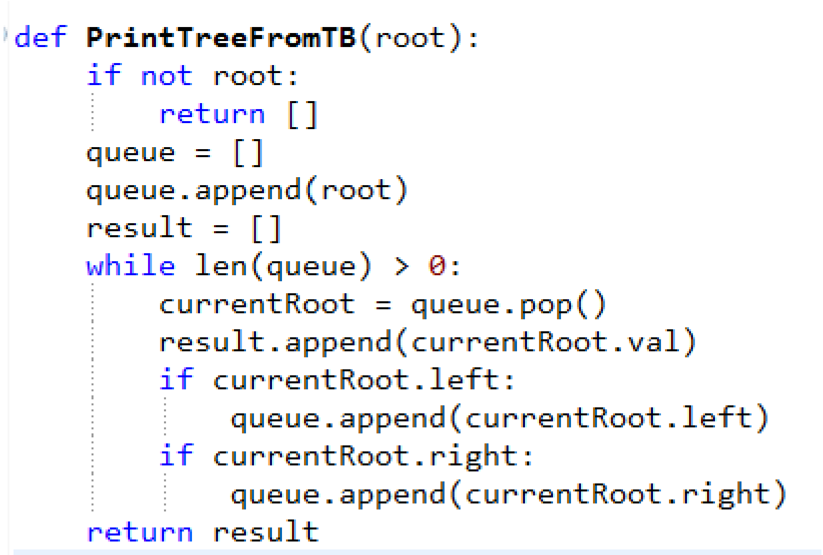
二叉树专题

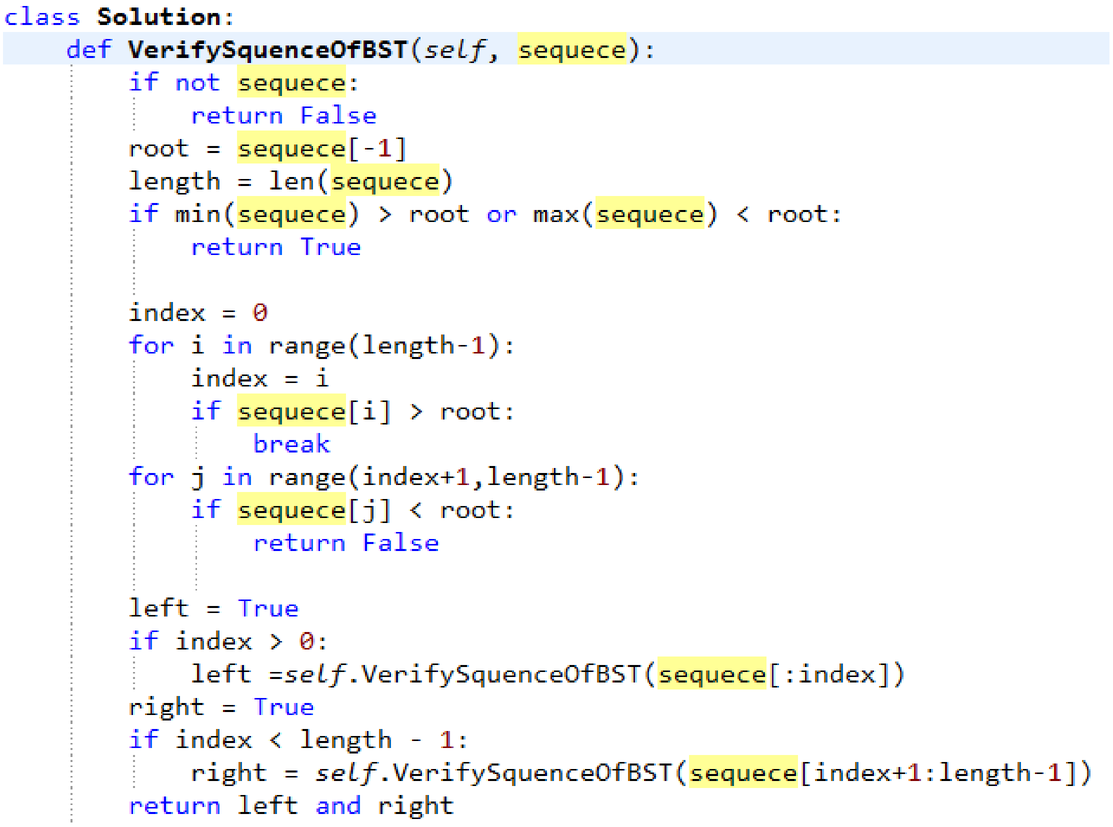
1、从上到下打印二叉树



```
def PrintTreeFromTB(root):
    if not root:
        return []
    queue = []
    queue.append(root)
    result = []
    while len(queue) > 0:
        currentRoot = queue.pop()
        result.append(currentRoot.val)
        if currentRoot.left:
            queue.append(currentRoot.left)
        if currentRoot.right:
            queue.append(currentRoot.right)
    return result
```

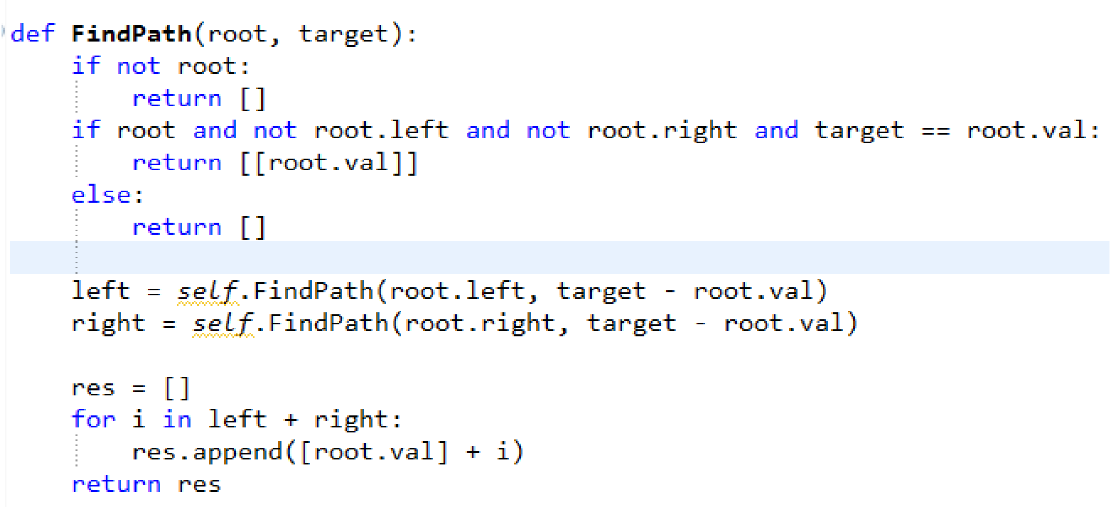
2、判断一个数组是否为某二叉搜索树的后续遍历结果。

思路：首先最后一个元素是二叉树的根节点，然后左子树都小于根节点，右子树都大于根节点。



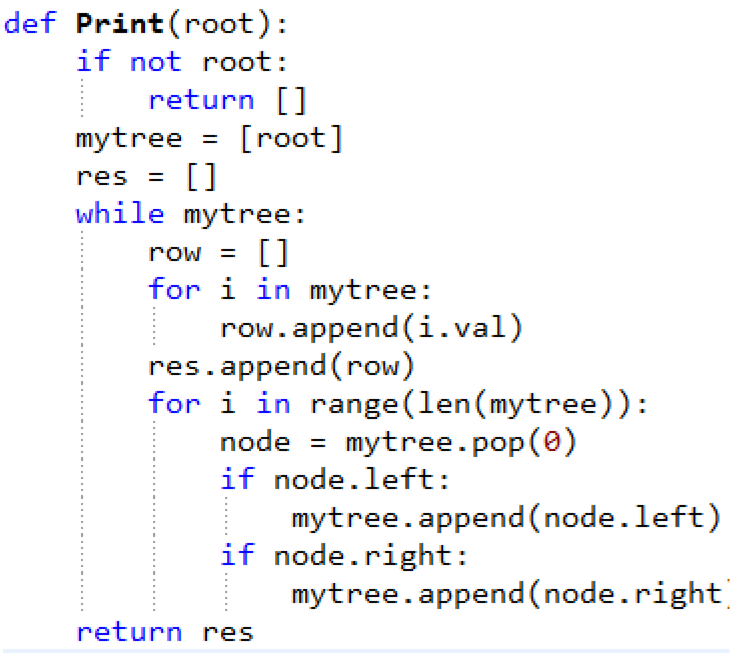
```
class Solution:
    def VerifySquenceOfBST(self, sequece):
        if not sequece:
            return False
        root = sequece[-1]
        length = len(sequece)
        if min(sequece) > root or max(sequece) < root:
            return True

        index = 0
        for i in range(length-1):
            index = i
            if sequece[i] > root:
                break
        for j in range(index+1,length-1):
            if sequece[j] < root:
                return False

        left = True
        if index > 0:
            left =self.VerifySquenceOfBST(sequece[:index])
        right = True
        if index < length - 1:
            right = self.VerifySquenceOfBST(sequece[index+1:length-1])
        return left and right
```

3、二叉树中和为某一值的路径



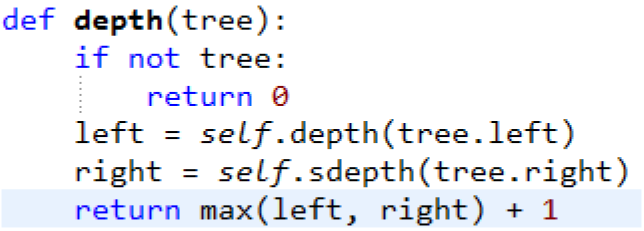
```python
def FindPath(root, target):
    if not root:
        return []
    if root and not root.left and not root.right and target == root.val:
        return [[root.val]]
    else:
        return []

    left = self.FindPath(root.left, target - root.val)
    right = self.FindPath(root.right, target - root.val)

    res = []
    for i in left + right:
        res.append([root.val] + i)
    return res
```

4、多行打印二叉树，要求每一层在一行



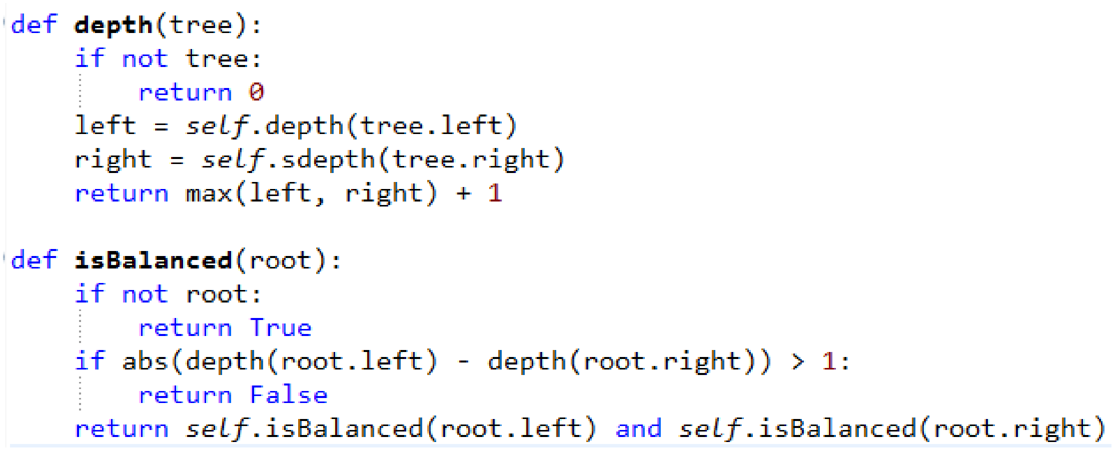
```python
def Print(root):
    if not root:
        return []
    mytree = [root]
    res = []
    while mytree:
        row = []
        for i in mytree:
            row.append(i.val)
        res.append(row)
        for i in range(len(mytree)):
            node = mytree.pop(0)
            if node.left:
                mytree.append(node.left)
            if node.right:
                mytree.append(node.right)
    return res
```

5、二叉树的深度



```python
def depth(tree):
    if not tree:
        return 0
    left = self.depth(tree.left)
    right = self.sdepth(tree.right)
    return max(left, right) + 1
```

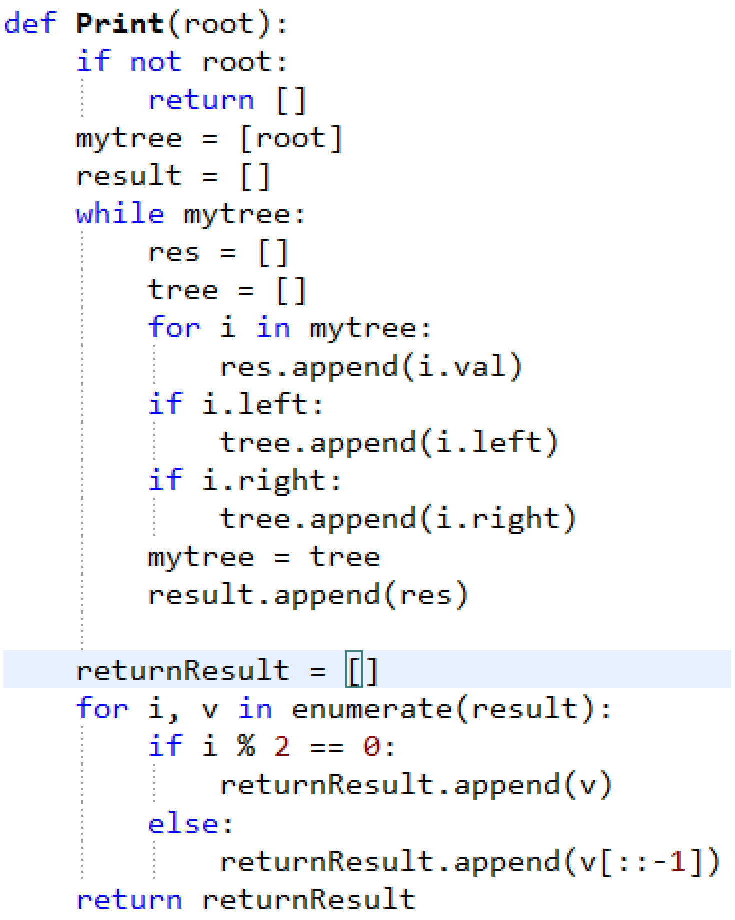
6、判断是否为平衡二叉树



```python
def depth(tree):
    if not tree:
        return 0
    left = self.depth(tree.left)
    right = self.sdepth(tree.right)
    return max(left, right) + 1

def isBalanced(root):
    if not root:
        return True
    if abs(depth(root.left) - depth(root.right)) > 1:
        return False
    return self.isBalanced(root.left) and self.isBalanced(root.right)
```

6、之字形打印二叉树

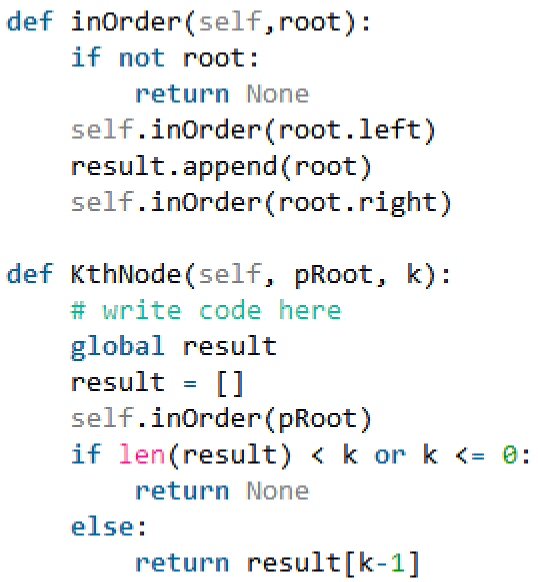
思路：加一个判断条件，奇数层和偶数层



```python
def Print(root):
    if not root:
        return []
    mytree = [root]
    result = []
    while mytree:
        res = []
        tree = []
        for i in mytree:
            res.append(i.val)
            if i.left:
                tree.append(i.left)
            if i.right:
                tree.append(i.right)
        mytree = tree
        result.append(res)

    returnResult = []
    for i, v in enumerate(result):
        if i % 2 == 0:
            returnResult.append(v)
        else:
            returnResult.append(v[::-1])
    return returnResult
```

7、二叉搜索树的第 k 个结点

思路：中序遍历输出一个序列，然后找到序列中第 k 个数即可。



```python
def inOrder(self,root):
    if not root:
        return None
    self.inOrder(root.left)
    result.append(root)
    self.inOrder(root.right)

def KthNode(self, pRoot, k):
    # write code here
    global result
    result = []
    self.inOrder(pRoot)
    if len(result) < k or k <= 0:
        return None
    else:
        return result[k-1]
```

数组专题

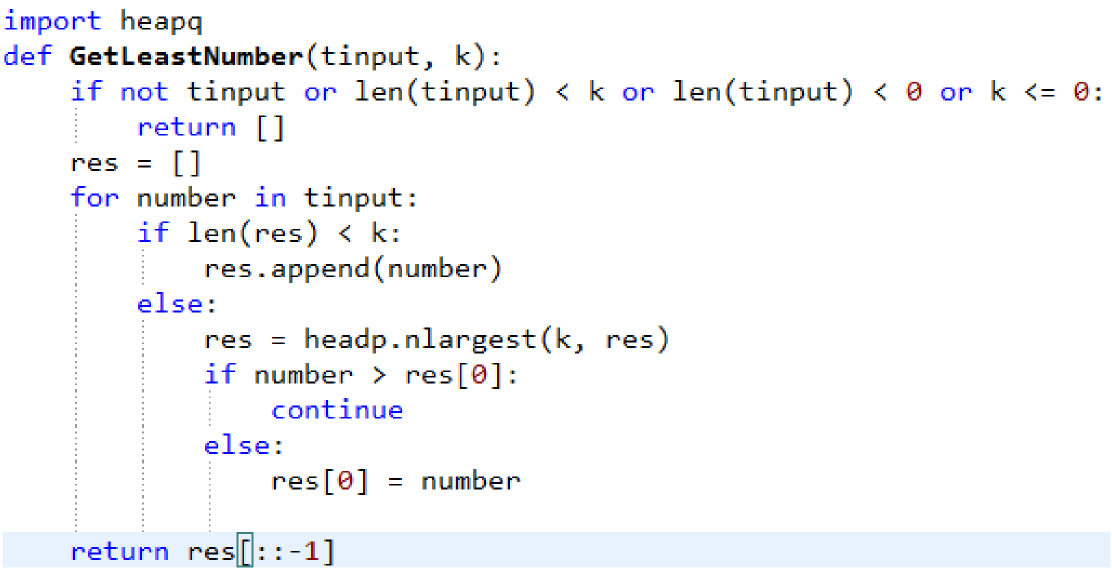
1、数组中出现次数超过一半的数字

(1) 第一种利用 python 的 collections.Counter 处理，然后在利用 items()：

(2) 第二种排序，处理 mid 应该是该数字，然后遍历对该数字出现次数计数：

2、最小的 k 个数

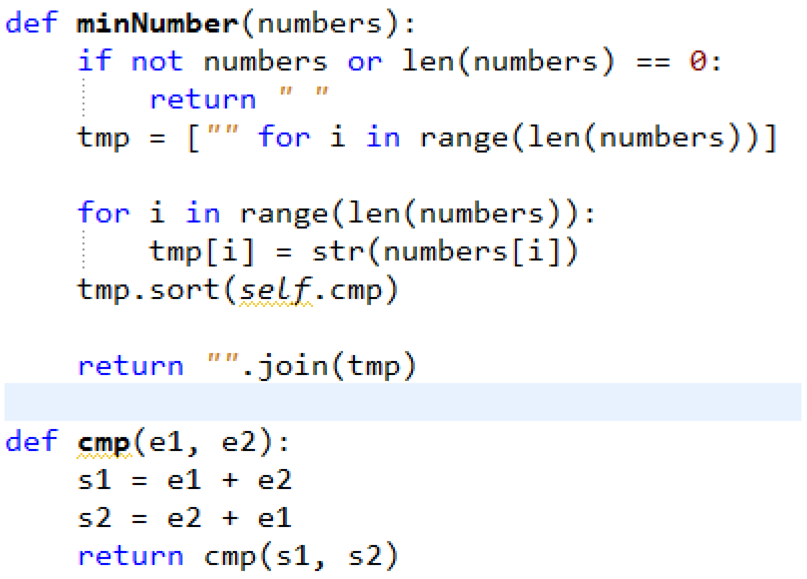
思路：构建最大堆；首先把数组前 k 个数字构建一个最大堆，然后从第 k+1 个数字开始遍历数组，如果遍历到的元素小于堆顶的数字，那么久将换两个数字，重新构造堆，继续遍历，最后剩下的堆就是最小的 k 个数，时间复杂度 O(nlog k)。



```python
import heapq
def GetLeastNumber(tinput, k):
    if not tinput or len(tinput) < k or len(tinput) < 0 or k <= 0:
        return []
    res = []
    for number in tinput:
        if len(res) < k:
            res.append(number)
        else:
            res = headp.nlargest(k, res)
            if number > res[0]:
                continue
            else:
                res[0] = number

    return res[::-1]
```

3、把数组排成最小的数

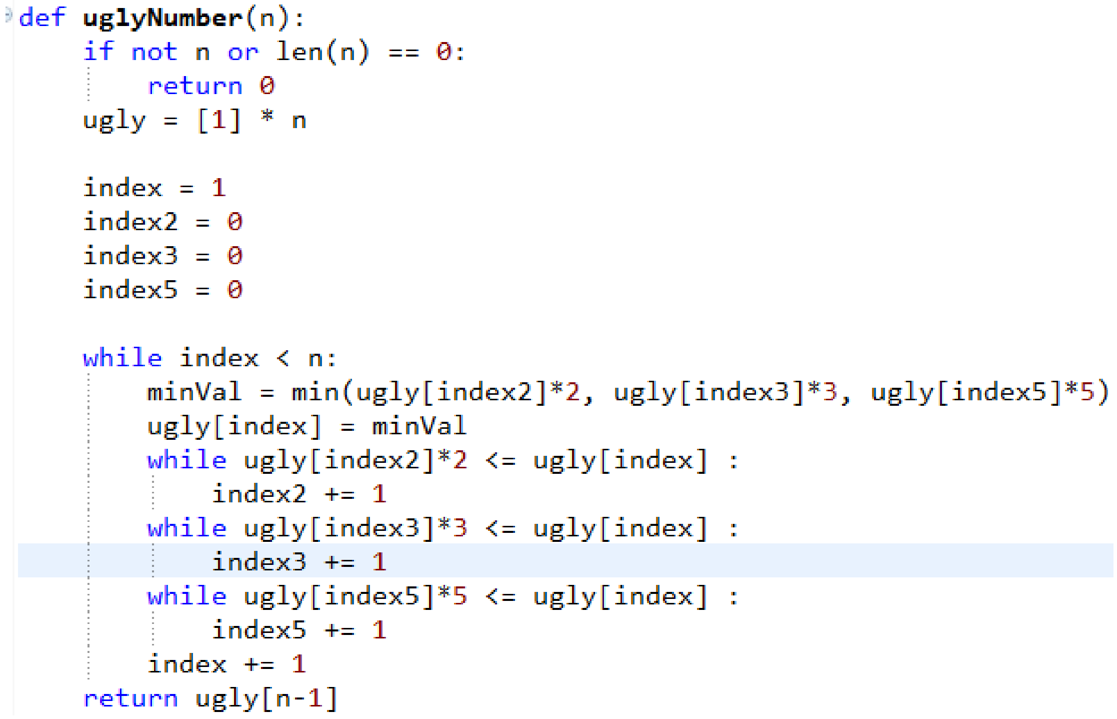
要把数字转成字符形式



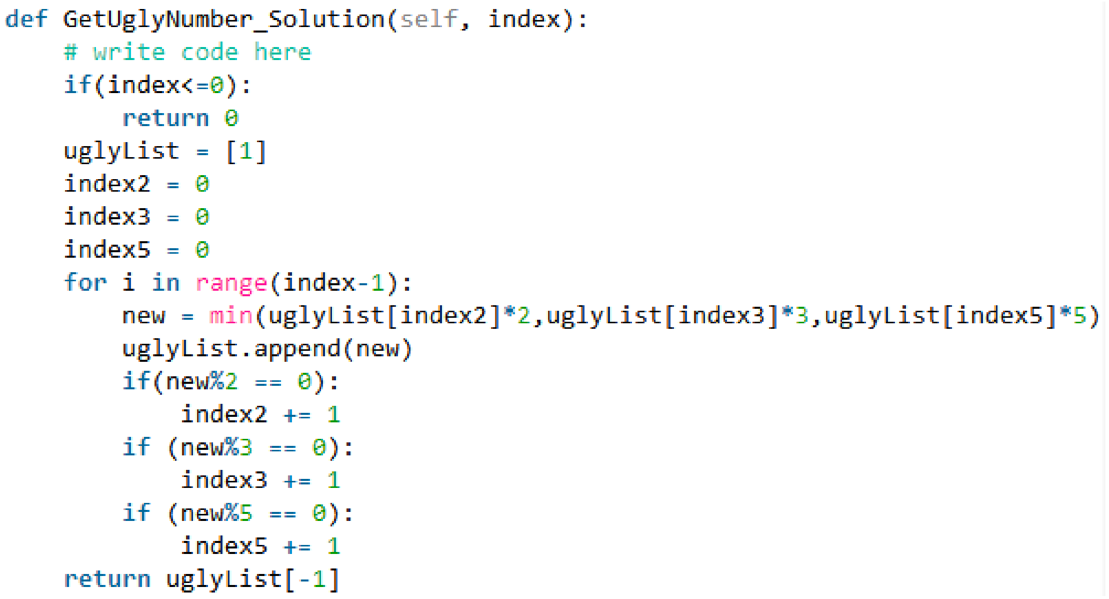
```python
def minNumber(numbers):
    if not numbers or len(numbers) == 0:
        return " "
    tmp = ["" for i in range(len(numbers))]

    for i in range(len(numbers)):
        tmp[i] = str(numbers[i])
    tmp.sort(self.cmp)

    return "".join(tmp)

def cmp(e1, e2):
    s1 = e1 + e2
    s2 = e2 + e1
    return cmp(s1, s2)
```

4、丑数

习惯上只包含 2、3、5 的数称为丑数，例如：6 和 8 都是丑数，习惯上我们把 1 当做第一个丑数：

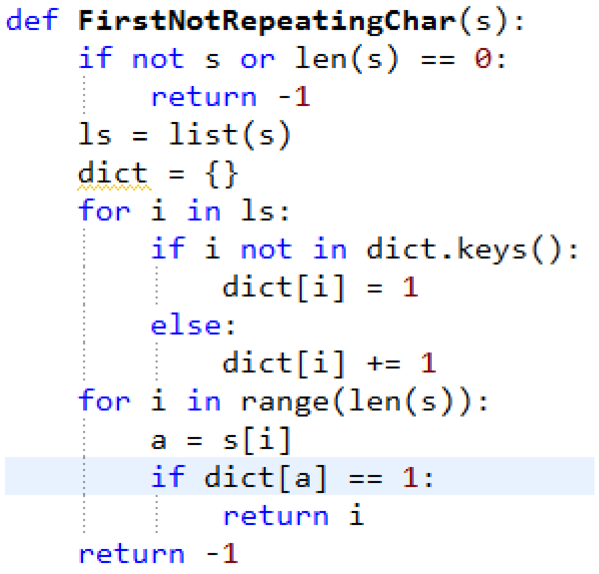


```python
def uglyNumber(n):
    if not n or len(n) == 0:
        return 0
    ugly = [1] * n

    index = 1
    index2 = 0
    index3 = 0
    index5 = 0

    while index < n:
        minVal = min(ugly[index2]*2, ugly[index3]*3, ugly[index5]*5)
        ugly[index] = minVal
        while ugly[index2]*2 <= ugly[index] :
            index2 += 1
        while ugly[index3]*3 <= ugly[index] :
            index3 += 1
        while ugly[index5]*5 <= ugly[index] :
            index5 += 1
        index += 1
    return ugly[n-1]
```



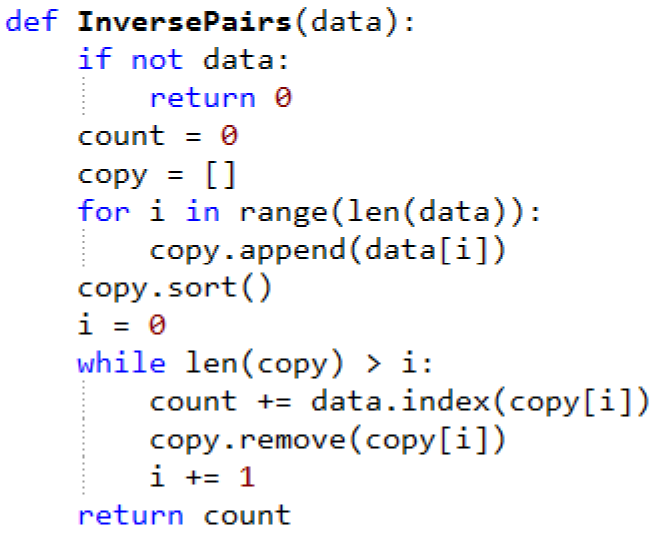
```
def GetUglyNumber_Solution(self, index):
    # write code here
    if(index<=0):
        return 0
    uglyList = [1]
    index2 = 0
    index3 = 0
    index5 = 0
    for i in range(index-1):
        new = min(uglyList[index2]*2,uglyList[index3]*3,uglyList[index5]*5)
        uglyList.append(new)
        if(new%2 == 0):
            index2 += 1
        if (new%3 == 0):
            index3 += 1
        if (new%5 == 0):
            index5 += 1
    return uglyList[-1]
```

5、第一个只出现一次的字符



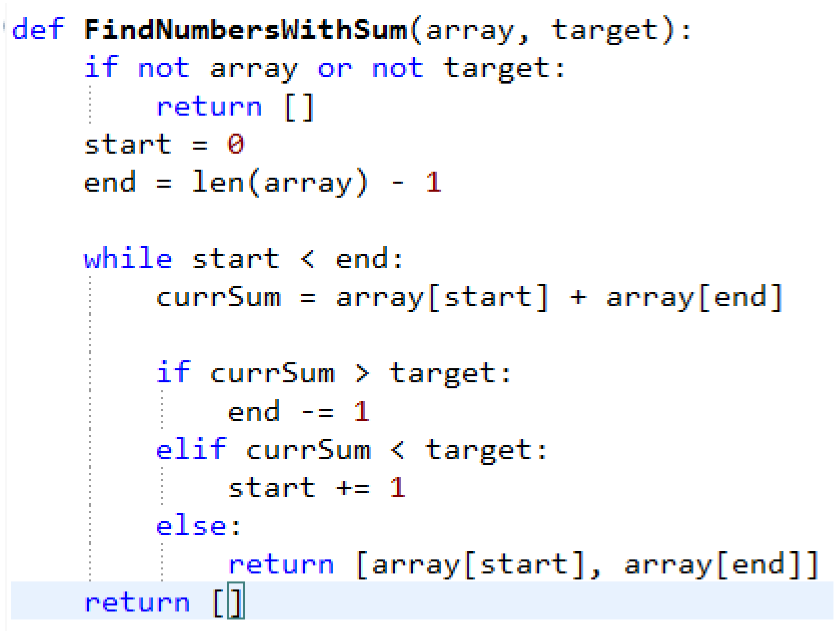
```
def FirstNotRepeatingChar(s):
    if not s or len(s) == 0:
        return -1
    ls = list(s)
    dict = {}
    for i in ls:
        if i not in dict.keys():
            dict[i] = 1
        else:
            dict[i] += 1
    for i in range(len(s)):
        a = s[i]
        if dict[a] == 1:
            return i
    return -1
```

6、逆序数对（未解决）



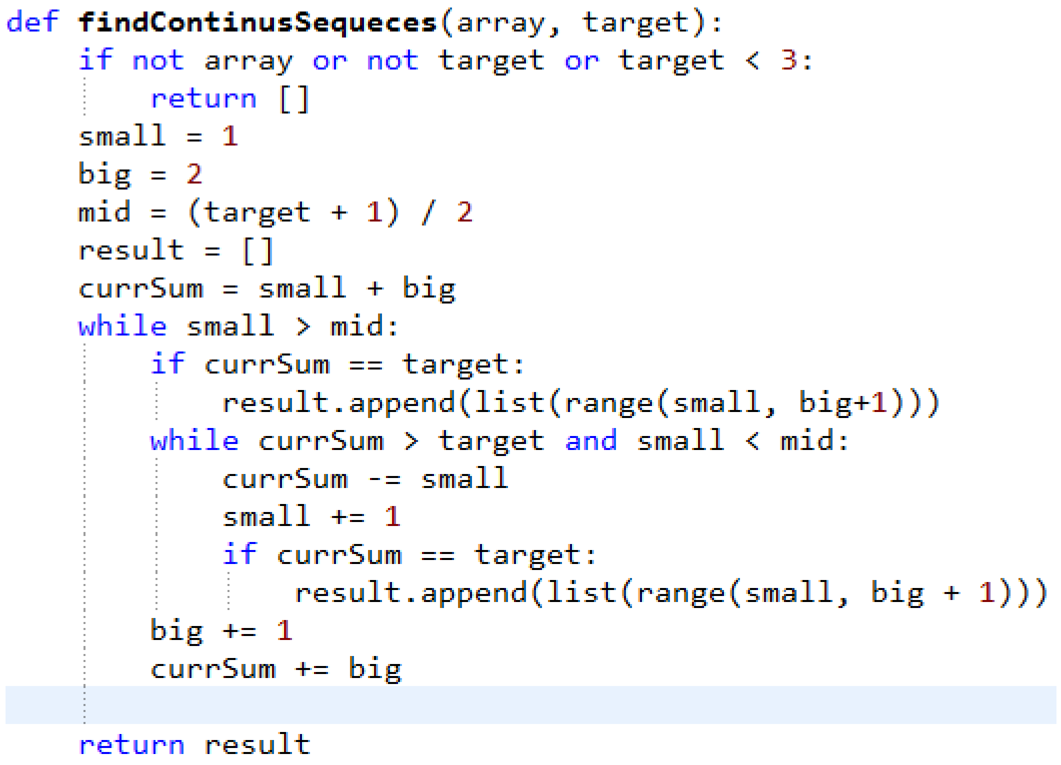
```
def InversePairs(data):
    if not data:
        return 0
    count = 0
    copy = []
    for i in range(len(data)):
        copy.append(data[i])
    copy.sort()
    i = 0
    while len(copy) > i:
        count += data.index(copy[i])
        copy.remove(copy[i])
        i += 1
    return count
```

7、和为 s 的两个数字



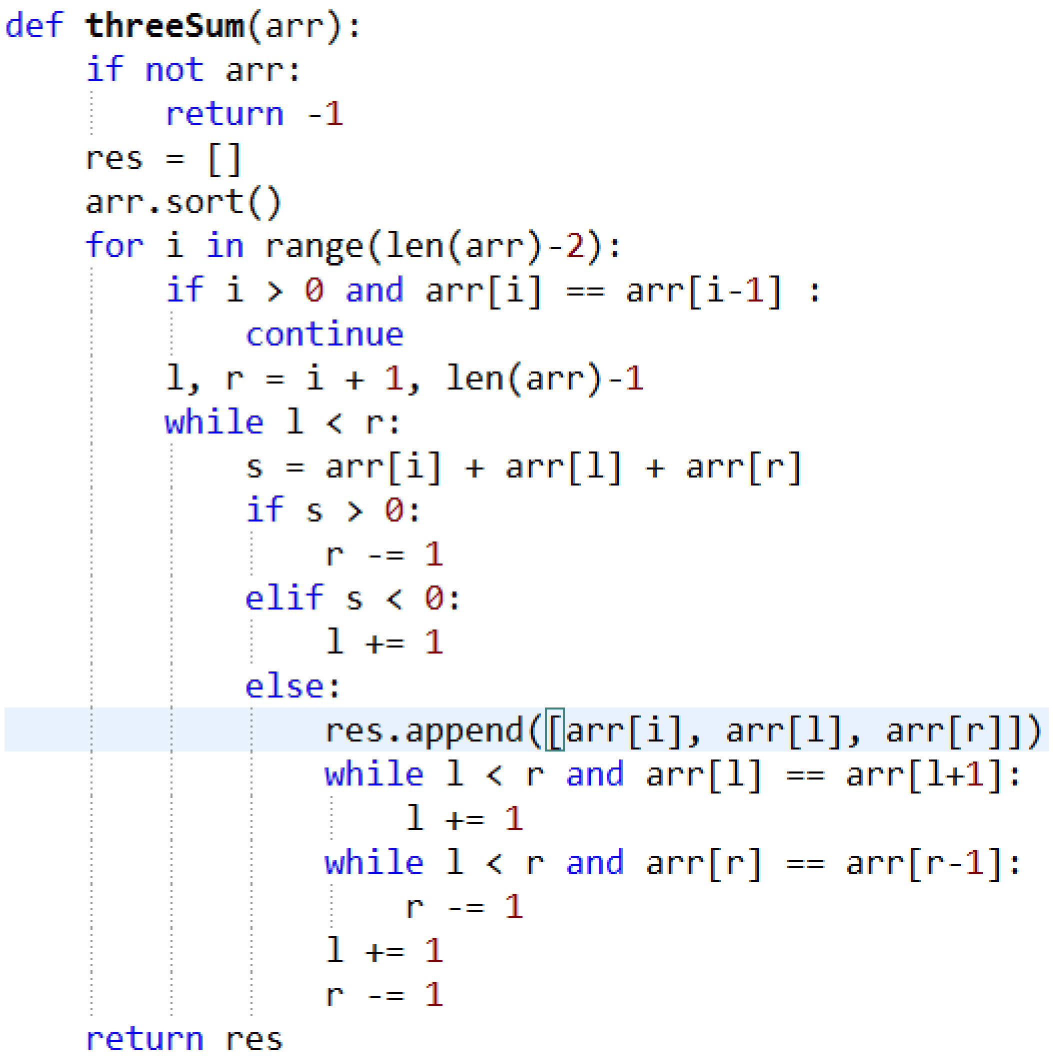
```python
def FindNumbersWithSum(array, target):
    if not array or not target:
        return []
    start = 0
    end = len(array) - 1

    while start < end:
        currSum = array[start] + array[end]

        if currSum > target:
            end -= 1
        elif currSum < target:
            start += 1
        else:
            return [array[start], array[end]]
    return []
```

8、和为 s 的连续正数序列



```python
def findContinusSequeces(array, target):
    if not array or not target or target < 3:
        return []
    small = 1
    big = 2
    mid = (target + 1) / 2
    result = []
    currSum = small + big
    while small > mid:
        if currSum == target:
            result.append(list(range(small, big+1)))
        while currSum > target and small < mid:
            currSum -= small
            small += 1
            if currSum == target:
                result.append(list(range(small, big + 1)))
        big += 1
        currSum += big

    return result
```

9、和为 s 的三个数

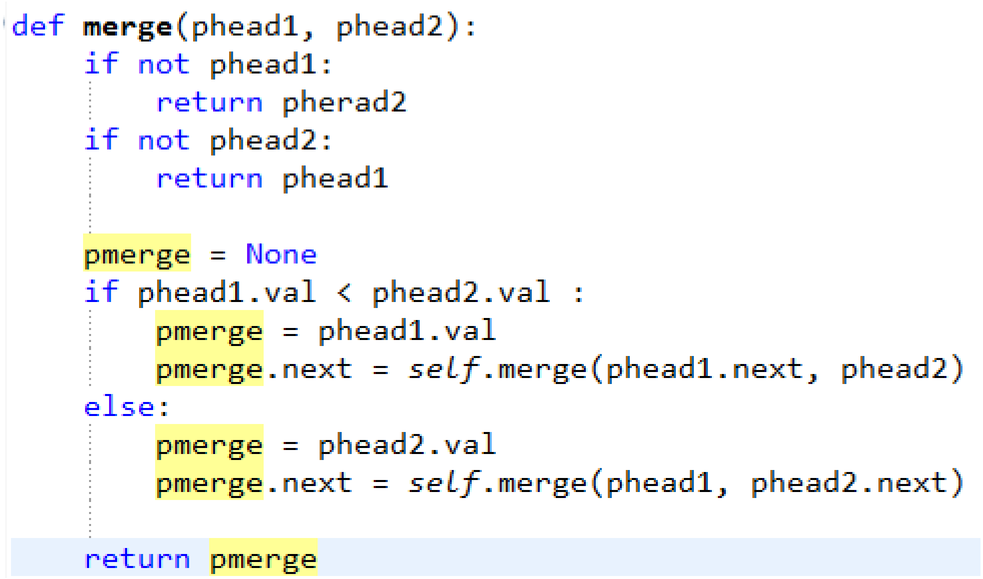


```
def threeSum(arr):
    if not arr:
        return -1
    res = []
    arr.sort()
    for i in range(len(arr)-2):
        if i > 0 and arr[i] == arr[i-1] :
            continue
        l, r = i + 1, len(arr)-1
        while l < r:
            s = arr[i] + arr[l] + arr[r]
            if s > 0:
                r -= 1
            elif s < 0:
                l += 1
            else:
                res.append([arr[i], arr[l], arr[r]])
                while l < r and arr[l] == arr[l+1]:
                    l += 1
                while l < r and arr[r] == arr[r-1]:
                    r -= 1
                l += 1
                r -= 1
    return res
```

10、4 个和为 s 的数相加

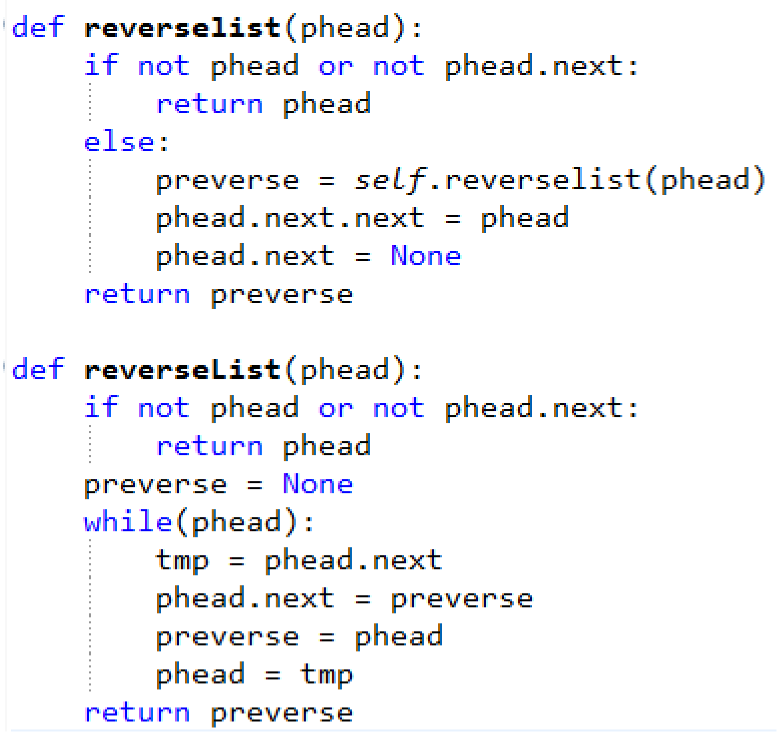
链表专题

1、合并两个排序的链表



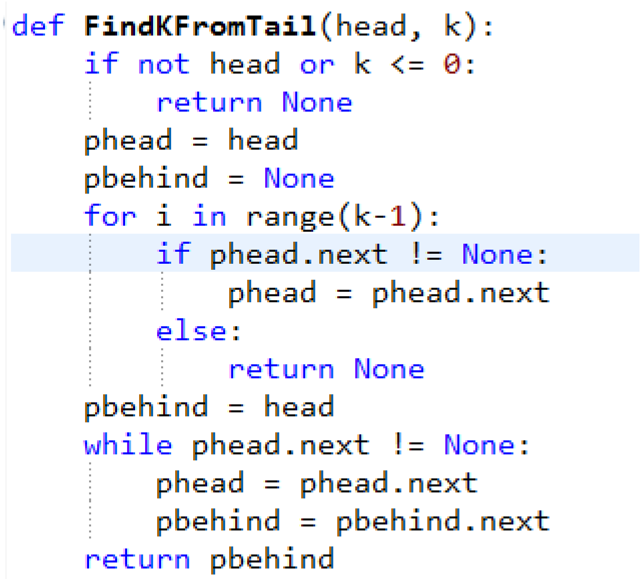
```
def merge(phead1, phead2):
    if not phead1:
        return pherad2
    if not phead2:
        return phead1

    pmerge = None
    if phead1.val < phead2.val :
        pmerge = phead1.val
        pmerge.next = self.merge(phead1.next, phead2)
    else:
        pmerge = phead2.val
        pmerge.next = self.merge(phead1, phead2.next)

    return pmerge
```

2、反转链表



```
def reverselist(phead):
    if not phead or not phead.next:
        return phead
    else:
        preverse = self.reverselist(phead)
        phead.next.next = phead
        phead.next = None
    return preverse

def reverseList(phead):
    if not phead or not phead.next:
        return phead
    preverse = None
    while(phead):
        tmp = phead.next
        phead.next = preverse
        preverse = phead
        phead = tmp
    return preverse
```

3、链表中的倒数第 k 个节点

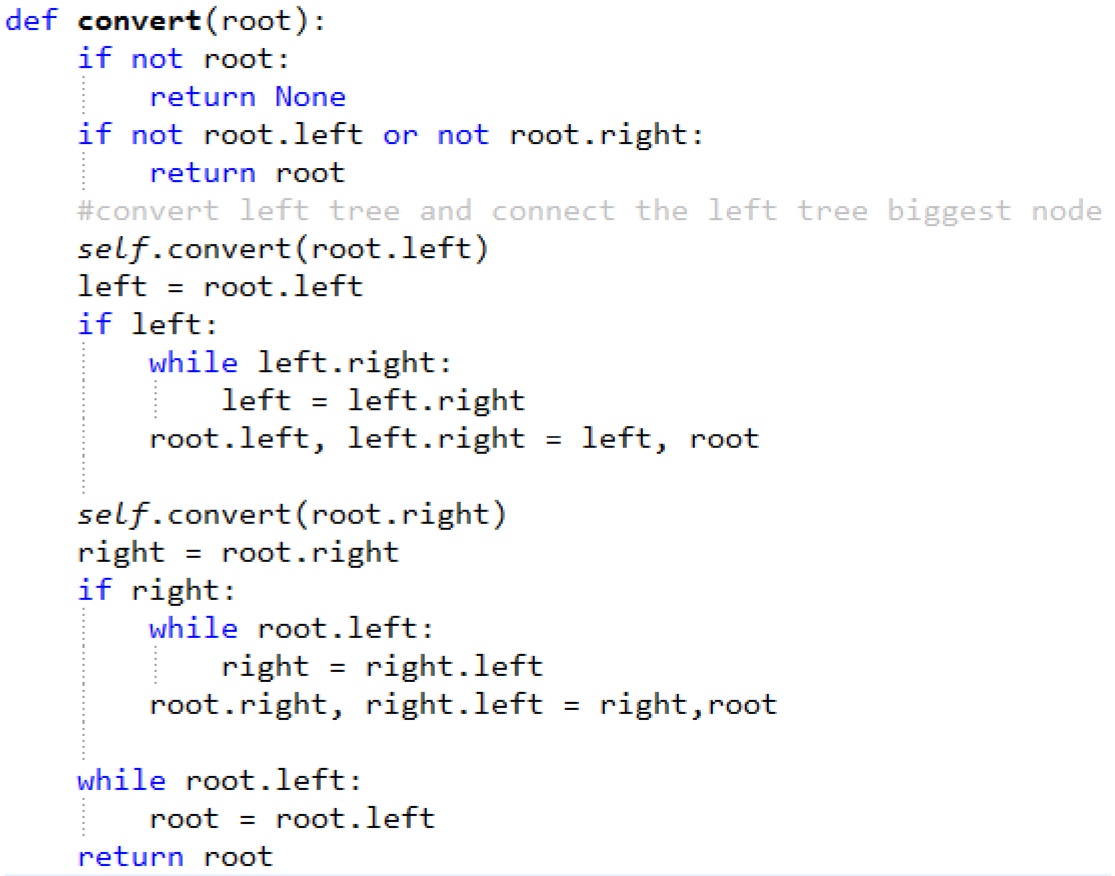


```
def FindKFromTail(head, k):
    if not head or k <= 0:
        return None
    phead = head
    pbehind = None
    for i in range(k-1):
        if phead.next != None:
            phead = phead.next
        else:
            return None
    pbehind = head
    while phead.next != None:
        phead = phead.next
        pbehind = pbehind.next
    return pbehind
```

4、二叉搜索树和双向链表

描述：输入一棵二叉搜索树，将该二叉树转换成一个排序的双向链表，要求不创建任何新的节点，只能调整树中指针的指向。

左子树小于根节点，右子树大于根节点，根节点左边连接左子树最大的结点，右边连接右子树最小的结点。

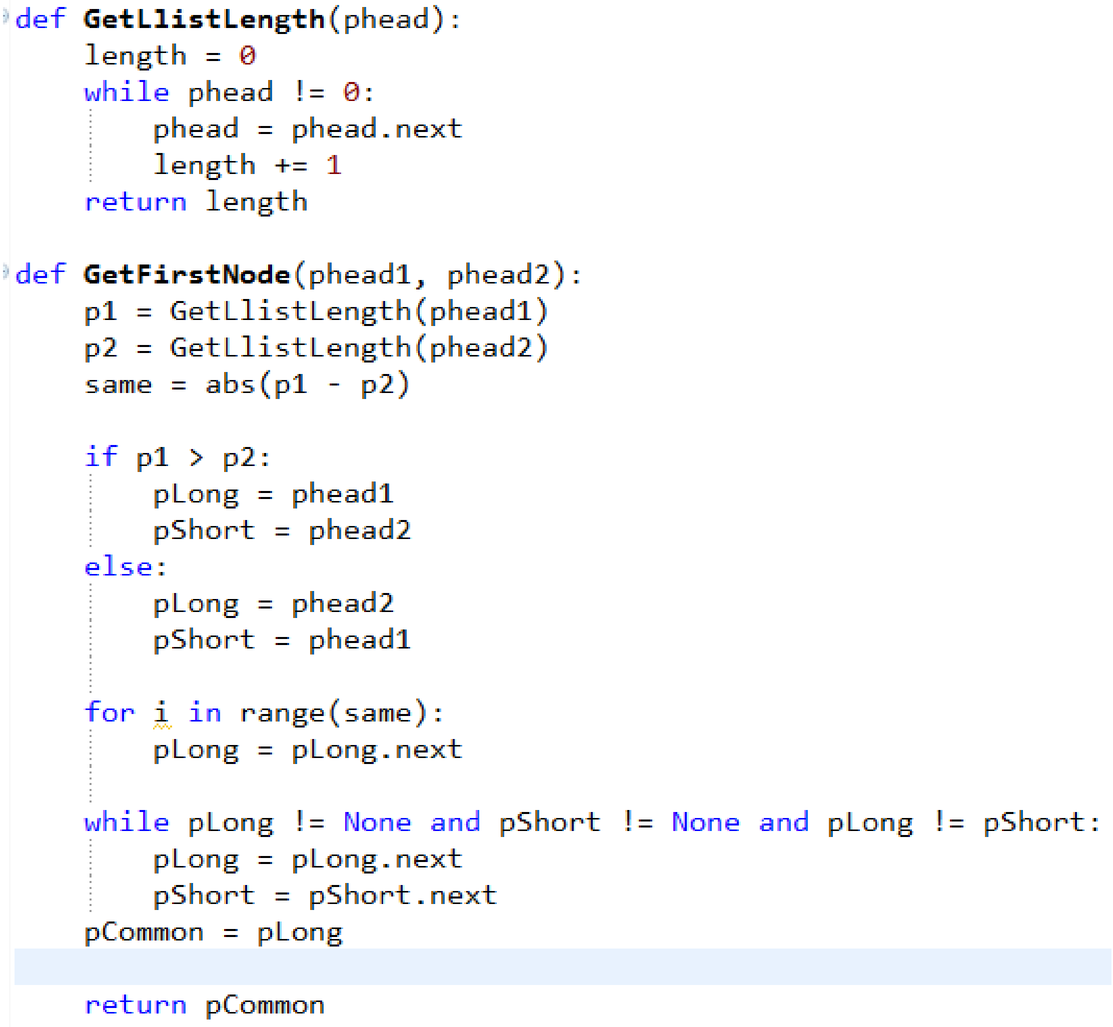


```
def convert(root):
    if not root:
        return None
    if not root.left or not root.right:
        return root
    #convert left tree and connect the left tree biggest node
    self.convert(root.left)
    left = root.left
    if left:
        while left.right:
            left = left.right
        root.left, left.right = left, root

    self.convert(root.right)
    right = root.right
    if right:
        while root.left:
            right = right.left
        root.right, right.left = right,root

    while root.left:
        root = root.left
    return root
```

5、两个链表的第一个公共结点

思路：首先依次遍历两个链表，记录两个链表的长度 m 和 n，如果 m＞n，那么我们就先让长度为 m 的链表走 m-n 个结点，然后两个链表同时遍历，当遍历到

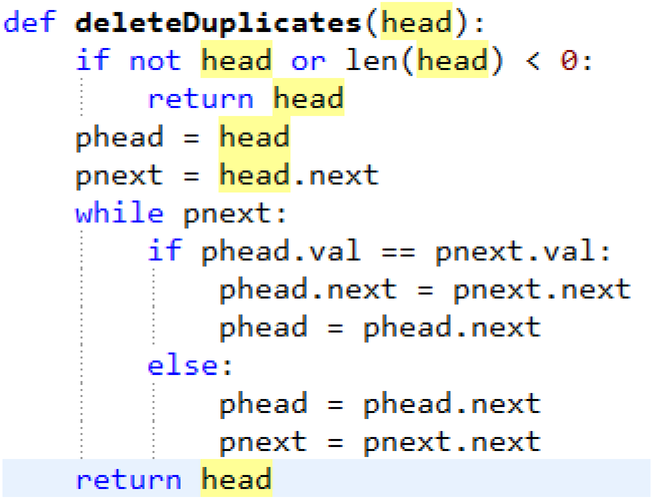
相同的结点的时候停止即可。对于 m < n，同理



```
def GetLlistLength(phead):
    length = 0
    while phead != 0:
        phead = phead.next
        length += 1
    return length

def GetFirstNode(phead1, phead2):
    p1 = GetLlistLength(phead1)
    p2 = GetLlistLength(phead2)
    same = abs(p1 - p2)

    if p1 > p2:
        pLong = phead1
        pShort = phead2
    else:
        pLong = phead2
        pShort = phead1

    for i in range(same):
        pLong = pLong.next

    while pLong != None and pShort != None and pLong != pShort:
        pLong = pLong.next
        pShort = pShort.next
    pCommon = pLong

    return pCommon
```

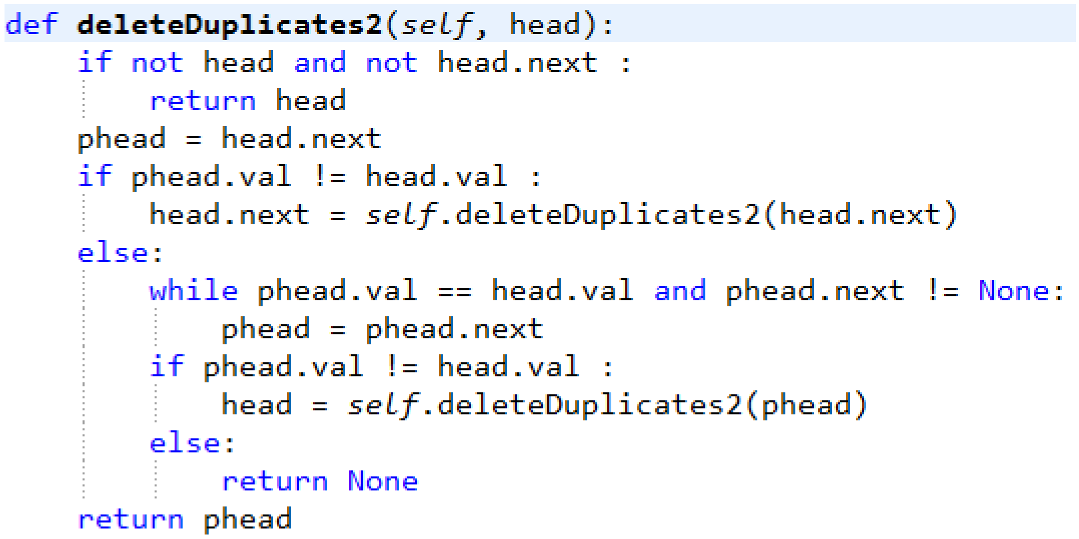
6、删除链表中的重复结点（保留重复结点）

思路：1->1->2；变成 1->2



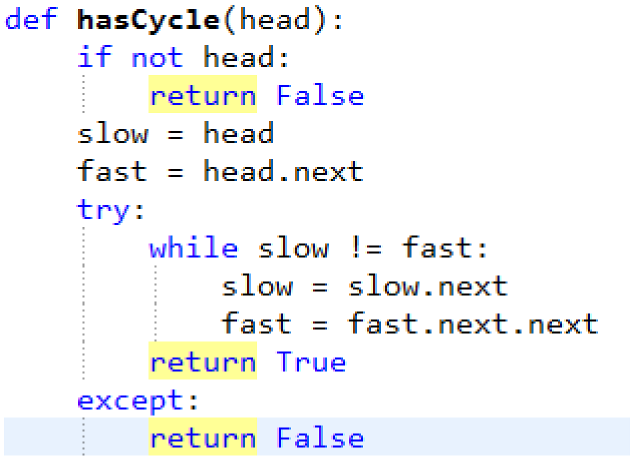
```
def deleteDuplicates(head):
    if not head or len(head) < 0:
        return head
    phead = head
    pnext = head.next
    while pnext:
        if phead.val == pnext.val:
            phead.next = pnext.next
            phead = phead.next
        else:
            phead = phead.next
            pnext = pnext.next
    return head
```

7、删除链表中的重复结点（不保留重复结点）



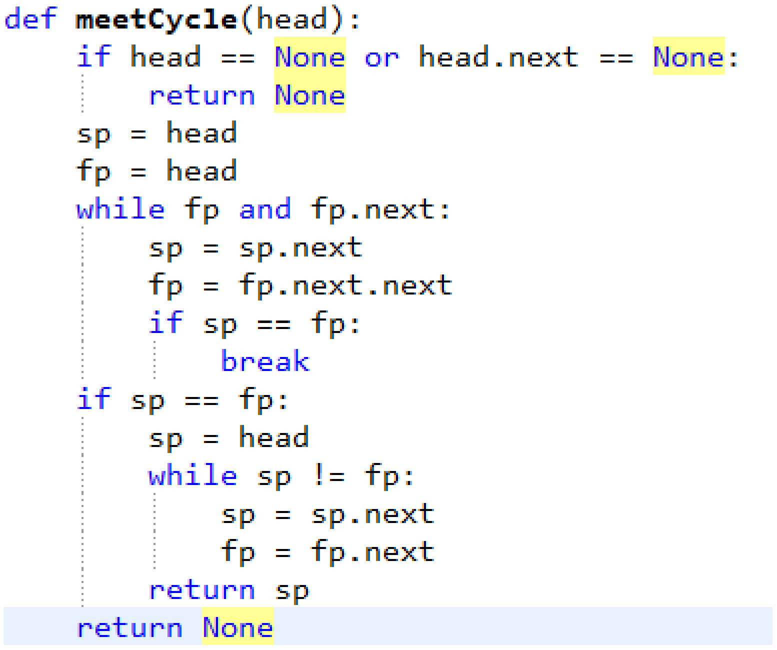
```
def deleteDuplicates2(self, head):
    if not head and not head.next :
        return head
    phead = head.next
    if phead.val != head.val :
        head.next = self.deleteDuplicates2(head.next)
    else:
        while phead.val == head.val and phead.next != None:
            phead = phead.next
        if phead.val != head.val :
            head = self.deleteDuplicates2(phead)
        else:
            return None
    return phead
```

8、判断链表是否存在环



```
def hasCycle(head):
    if not head:
        return False
    slow = head
    fast = head.next
    try:
        while slow != fast:
            slow = slow.next
            fast = fast.next.next
        return True
    except:
        return False
```

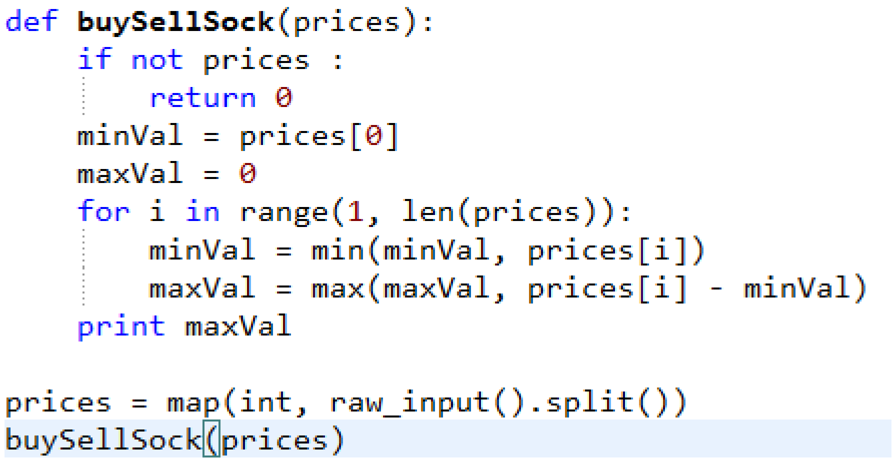
9、判断链表环入口（快慢指针实现）



```
def meetCycle(head):
    if head == None or head.next == None:
        return None
    sp = head
    fp = head
    while fp and fp.next:
        sp = sp.next
        fp = fp.next.next
        if sp == fp:
            break
    if sp == fp:
        sp = head
        while sp != fp:
            sp = sp.next
            fp = fp.next
        return sp
    return None
```

动态规划

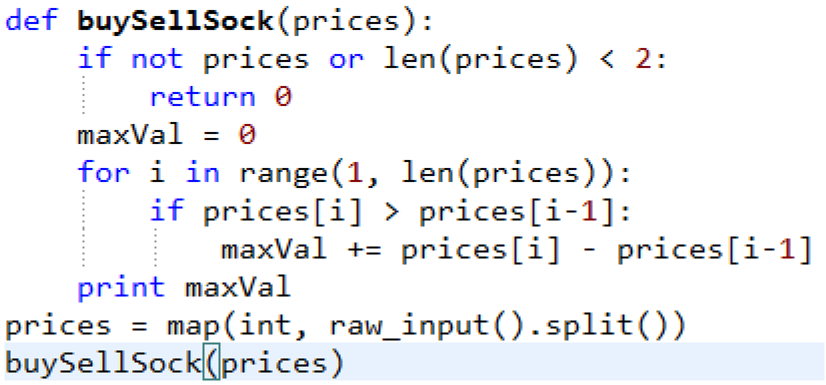
1、最好的时间买入和卖出股票（只可以交易一次）



```
def buySellSock(prices):
    if not prices :
        return 0
    minVal = prices[0]
    maxVal = 0
    for i in range(1, len(prices)):
        minVal = min(minVal, prices[i])
        maxVal = max(maxVal, prices[i] - minVal)
    print maxVal

prices = map(int, raw_input().split())
buySellSock(prices)
```

2、买入卖出股票的最大利润（允许多次交易）

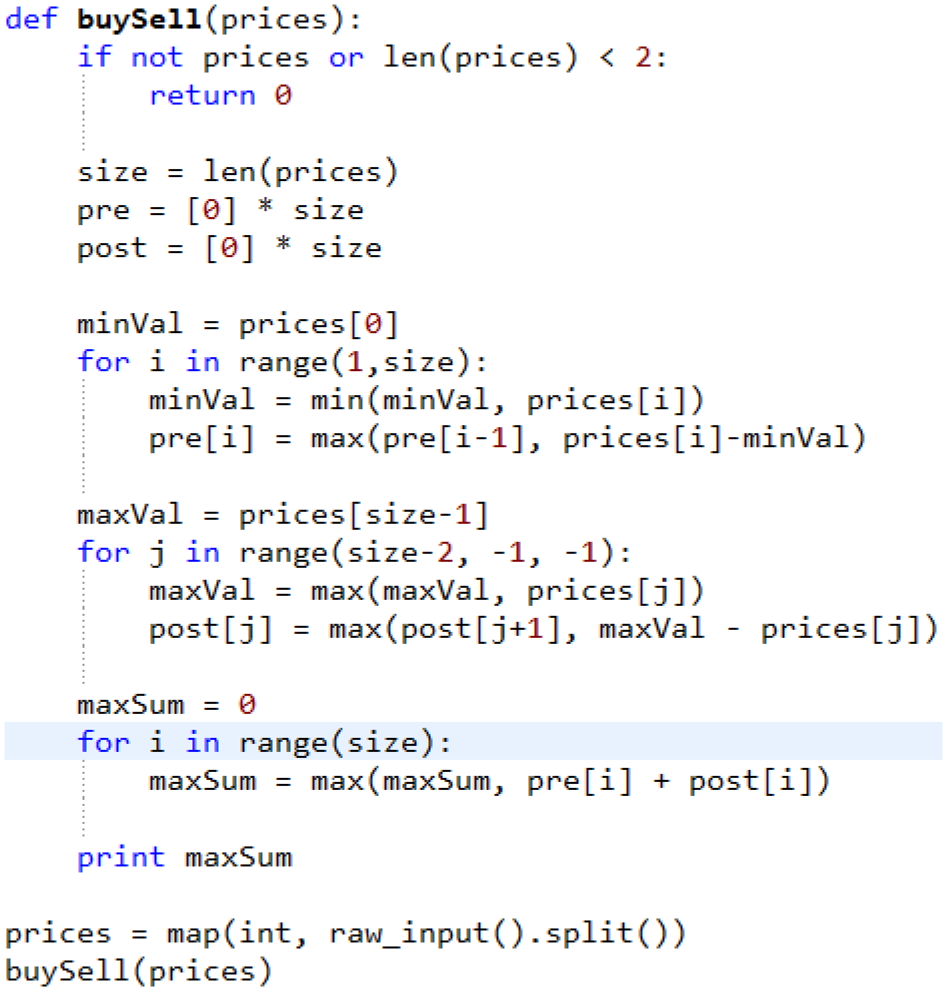
思路：判断前一个比后一个小，然后相减依次迭加就好了!!!



```
def buySellSock(prices):
    if not prices or len(prices) < 2:
        return 0
    maxVal = 0
    for i in range(1, len(prices)):
        if prices[i] > prices[i-1]:
            maxVal += prices[i] - prices[i-1]
    print maxVal
prices = map(int, raw_input().split())
buySellSock(prices)
```

3、买入卖出股票的最大利润（允许两次交易）

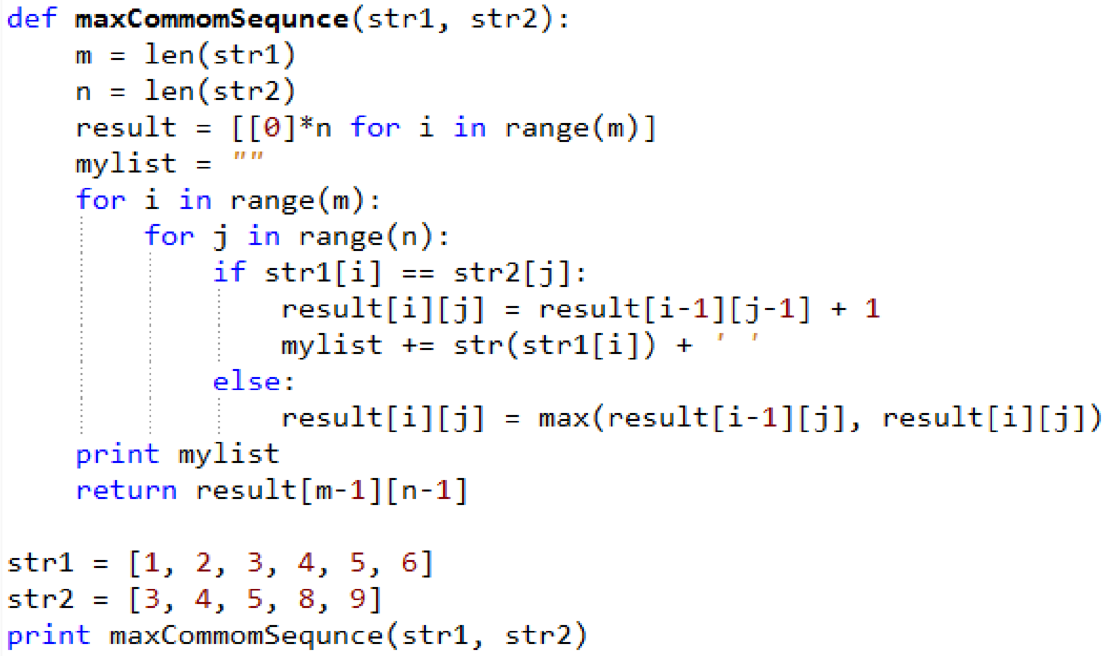
思路：找到一个结点 i， 判断 i 之前的最大利润和 i 之后的最大利润。



```
def buySell(prices):
    if not prices or len(prices) < 2:
        return 0

    size = len(prices)
    pre = [0] * size
    post = [0] * size

    minVal = prices[0]
    for i in range(1,size):
        minVal = min(minVal, prices[i])
        pre[i] = max(pre[i-1], prices[i]-minVal)

    maxVal = prices[size-1]
    for j in range(size-2, -1, -1):
        maxVal = max(maxVal, prices[j])
        post[j] = max(post[j+1], maxVal - prices[j])

    maxSum = 0
    for i in range(size):
        maxSum = max(maxSum, pre[i] + post[i])

    print maxSum

prices = map(int, raw_input().split())
buySell(prices)
```

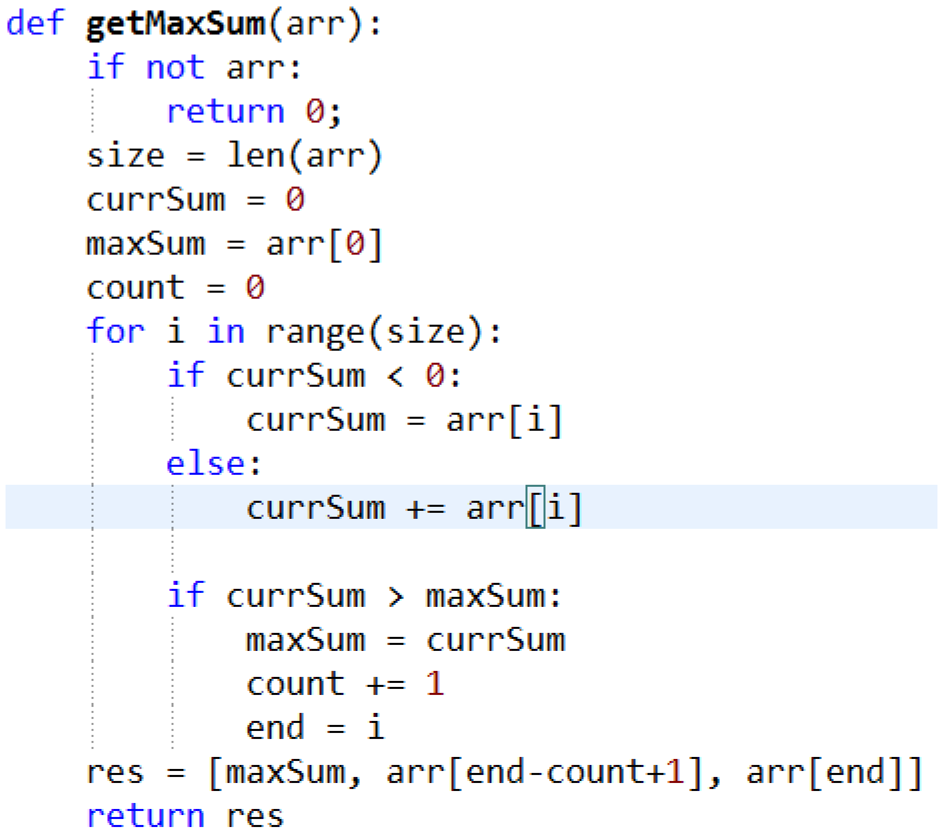
4、最长公共子序列

思路：计算两个序列的长度，利用动态规划的思想，新建一个 m*n 的数组，加入相等的话，dp[i][j]=dp[i-1][j-1]+1；否则 dp[i][j]=max(dp[i-1][j],dp[i[j-1])



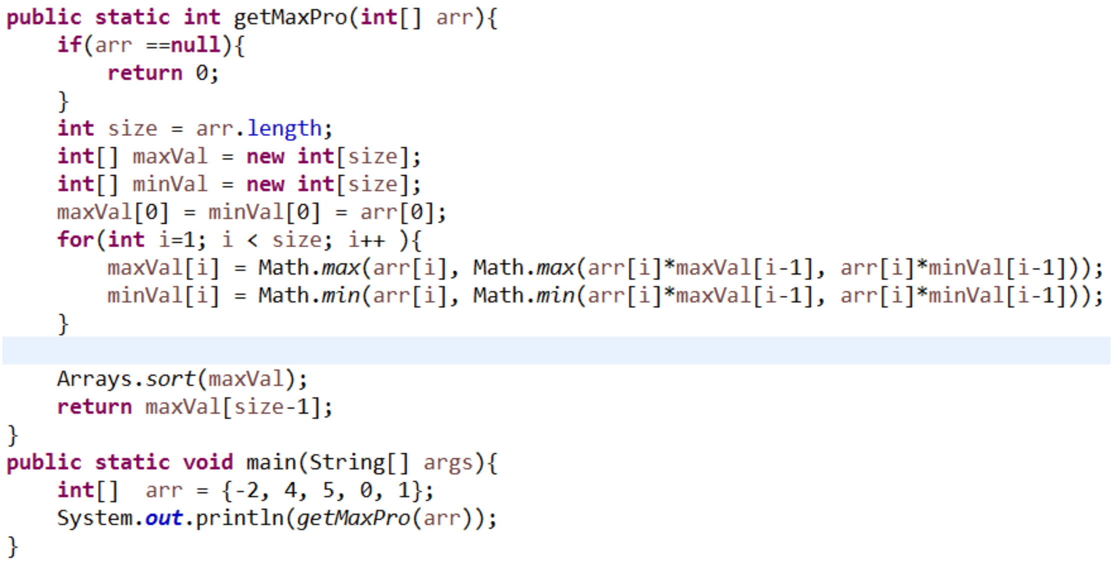
```
def maxCommomSequnce(str1, str2):
    m = len(str1)
    n = len(str2)
    result = [[0]*n for i in range(m)]
    mylist = ""
    for i in range(m):
        for j in range(n):
            if str1[i] == str2[j]:
                result[i][j] = result[i-1][j-1] + 1
                mylist += str(str1[i]) + ' '
            else:
                result[i][j] = max(result[i-1][j], result[i][j])
    print mylist
    return result[m-1][n-1]

str1 = [1, 2, 3, 4, 5, 6]
str2 = [3, 4, 5, 8, 9]
print maxCommomSequnce(str1, str2)
```

5、最大子序列和

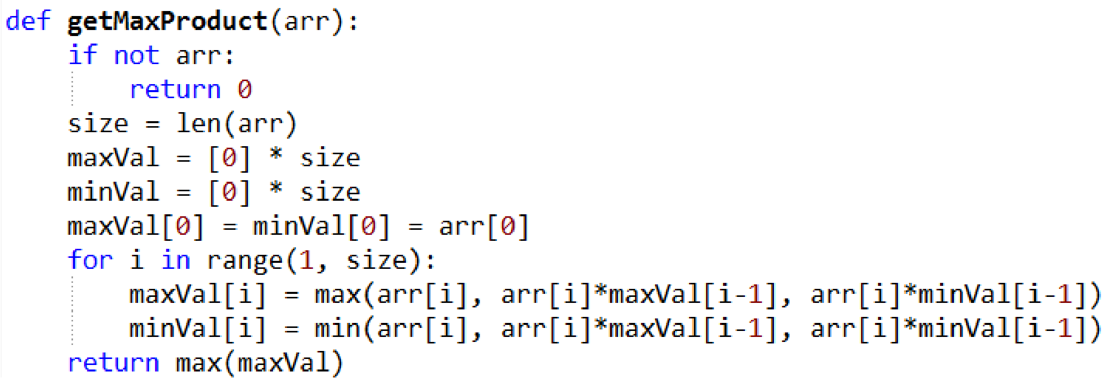


```python
def getMaxSum(arr):
    if not arr:
        return 0;
    size = len(arr)
    currSum = 0
    maxSum = arr[0]
    count = 0
    for i in range(size):
        if currSum < 0:
            currSum = arr[i]
        else:
            currSum += arr[i]

        if currSum > maxSum:
            maxSum = currSum
            count += 1
            end = i
    res = [maxSum, arr[end-count+1], arr[end]]
    return res
```

6、最大子序列乘积

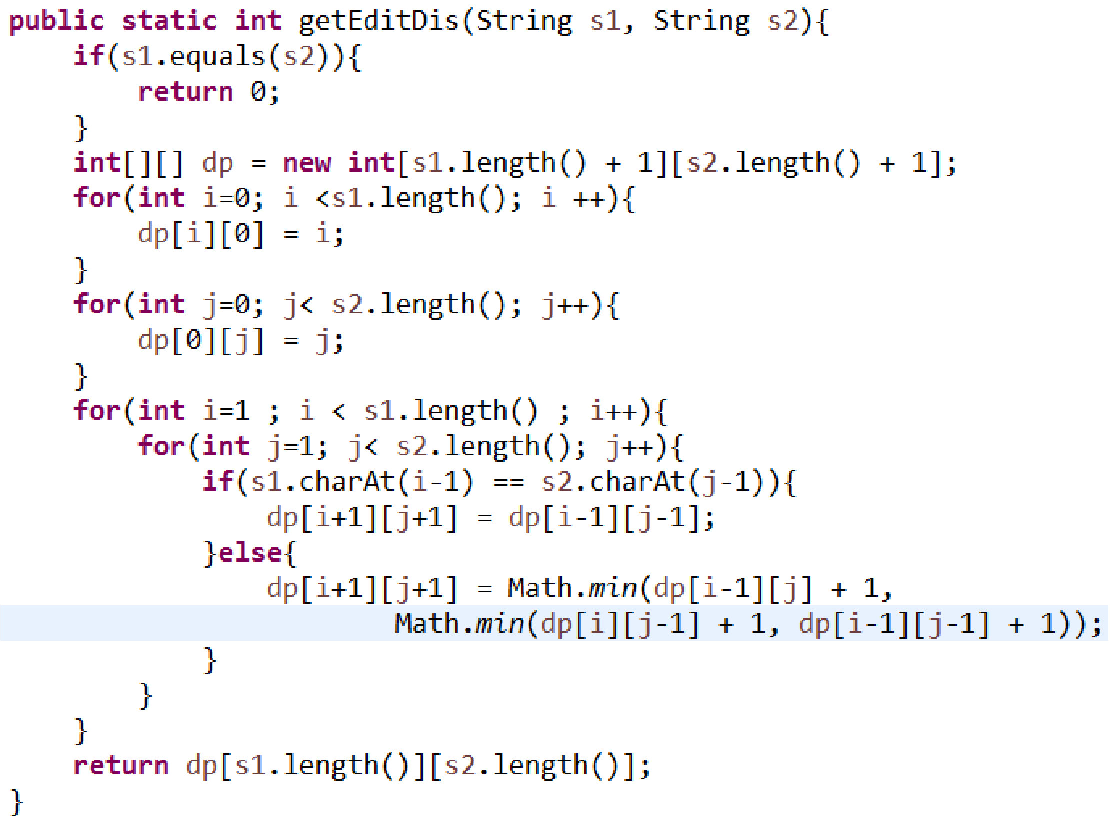


```java
public static int getMaxPro(int[] arr){
    if(arr ==null){
        return 0;
    }
    int size = arr.length;
    int[] maxVal = new int[size];
    int[] minVal = new int[size];
    maxVal[0] = minVal[0] = arr[0];
    for(int i=1; i < size; i++ ){
        maxVal[i] = Math.max(arr[i], Math.max(arr[i]*maxVal[i-1], arr[i]*minVal[i-1]));
        minVal[i] = Math.min(arr[i], Math.min(arr[i]*maxVal[i-1], arr[i]*minVal[i-1]));
    }

    Arrays.sort(maxVal);
    return maxVal[size-1];
}
public static void main(String[] args){
    int[]  arr = {-2, 4, 5, 0, 1};
    System.out.println(getMaxPro(arr));
}
```



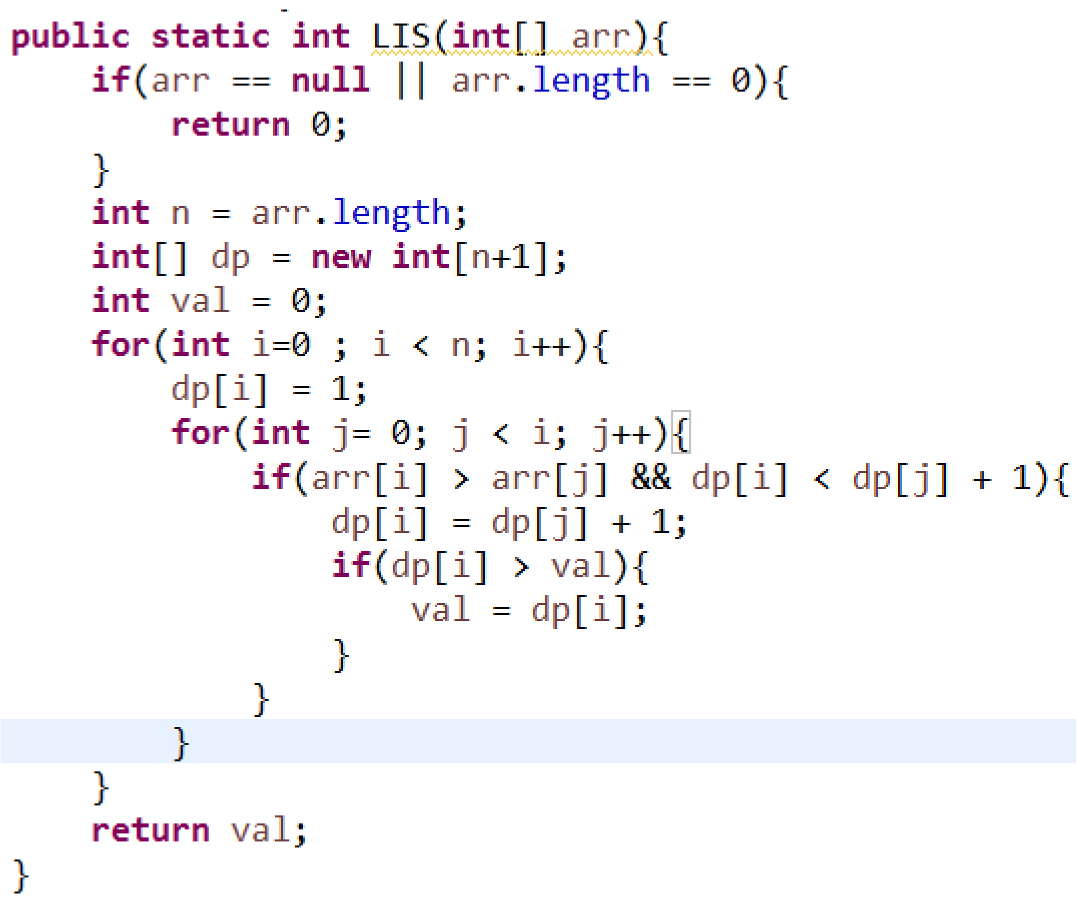
```python
def getMaxProduct(arr):
    if not arr:
        return 0
    size = len(arr)
    maxVal = [0] * size
    minVal = [0] * size
    maxVal[0] = minVal[0] = arr[0]
    for i in range(1, size):
        maxVal[i] = max(arr[i], arr[i]*maxVal[i-1], arr[i]*minVal[i-1])
        minVal[i] = min(arr[i], arr[i]*maxVal[i-1], arr[i]*minVal[i-1])
    return max(maxVal)
```

7、最小编辑距离



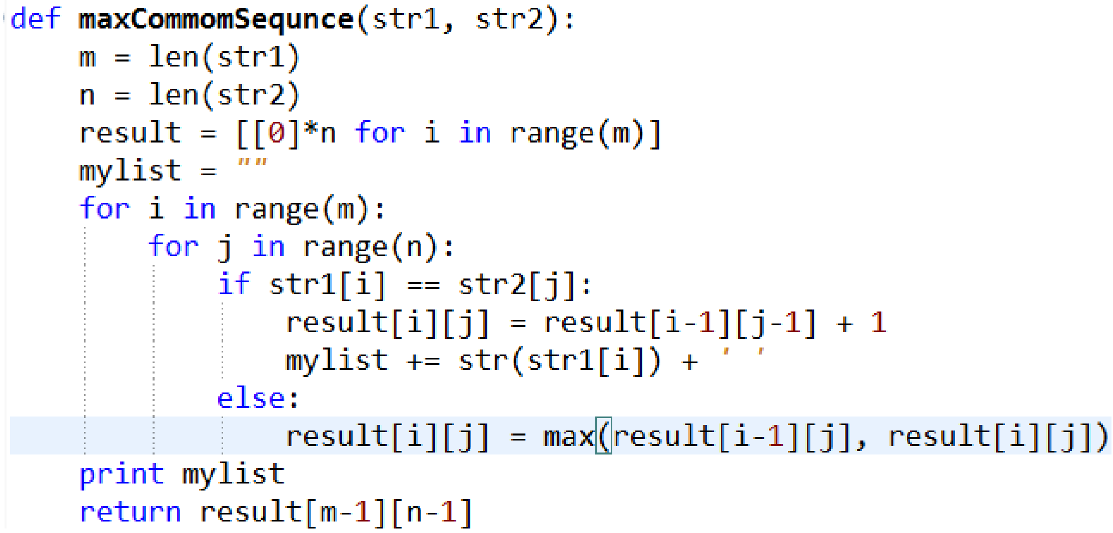
```java
public static int getEditDis(String s1, String s2){
    if(s1.equals(s2)){
        return 0;
    }
    int[][] dp = new int[s1.length() + 1][s2.length() + 1];
    for(int i=0; i <s1.length(); i ++){
        dp[i][0] = i;
    }
    for(int j=0; j< s2.length(); j++){
        dp[0][j] = j;
    }
    for(int i=1 ; i < s1.length() ; i++){
        for(int j=1; j< s2.length(); j++){
            if(s1.charAt(i-1) == s2.charAt(j-1)){
                dp[i+1][j+1] = dp[i-1][j-1];
            }else{
                dp[i+1][j+1] = Math.min(dp[i-1][j] + 1,
                        Math.min(dp[i][j-1] + 1, dp[i-1][j-1] + 1));
            }
        }
    }
    return dp[s1.length()][s2.length()];
}
```

8、最长递增子序列



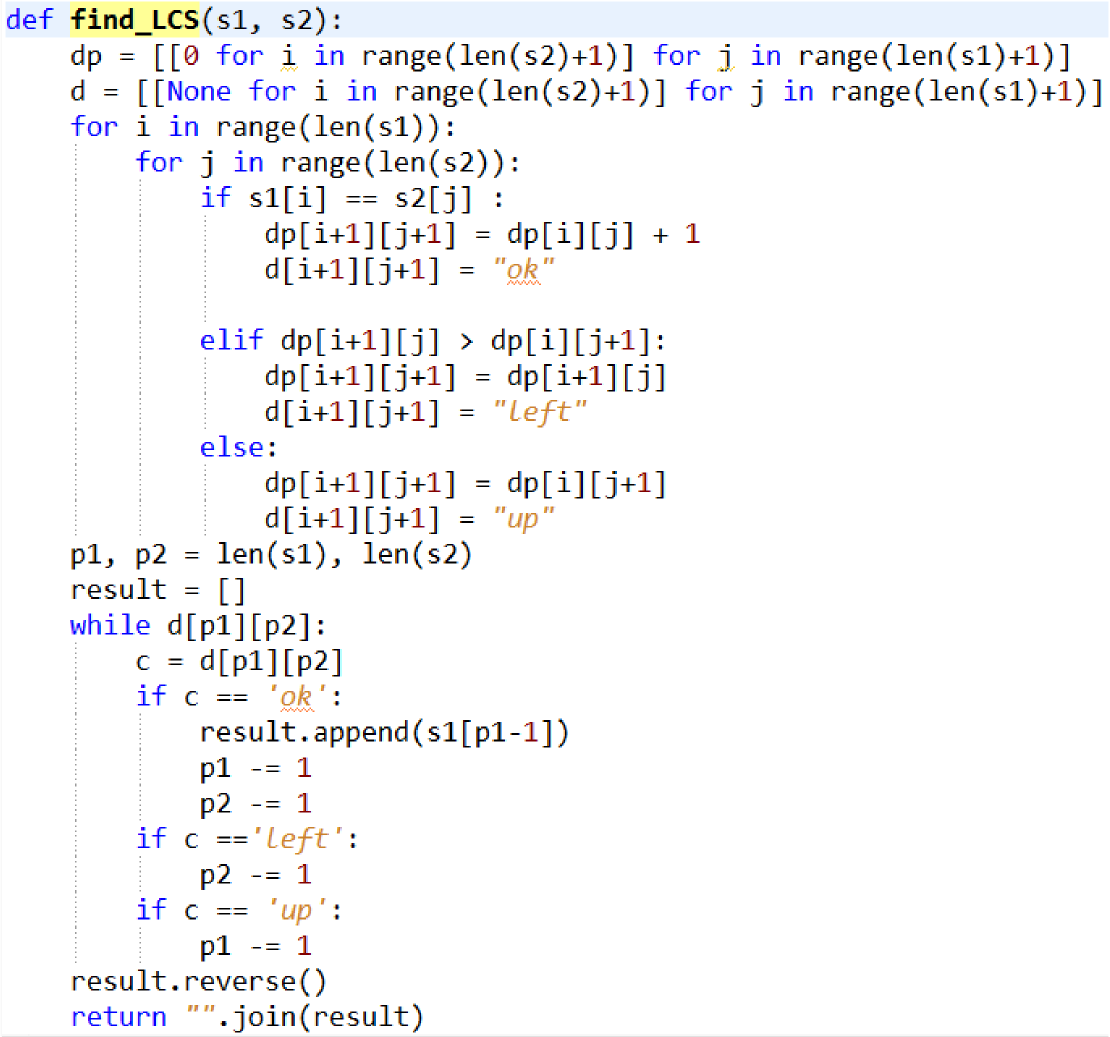
```java
public static int LIS(int[] arr){
    if(arr == null || arr.length == 0){
        return 0;
    }
    int n = arr.length;
    int[] dp = new int[n+1];
    int val = 0;
    for(int i=0 ; i < n; i++){
        dp[i] = 1;
        for(int j= 0; j < i; j++){
            if(arr[i] > arr[j] && dp[i] < dp[j] + 1){
                dp[i] = dp[j] + 1;
                if(dp[i] > val){
                    val = dp[i];
                }
            }
        }
    }
    return val;
}
```

9、最长公共子串



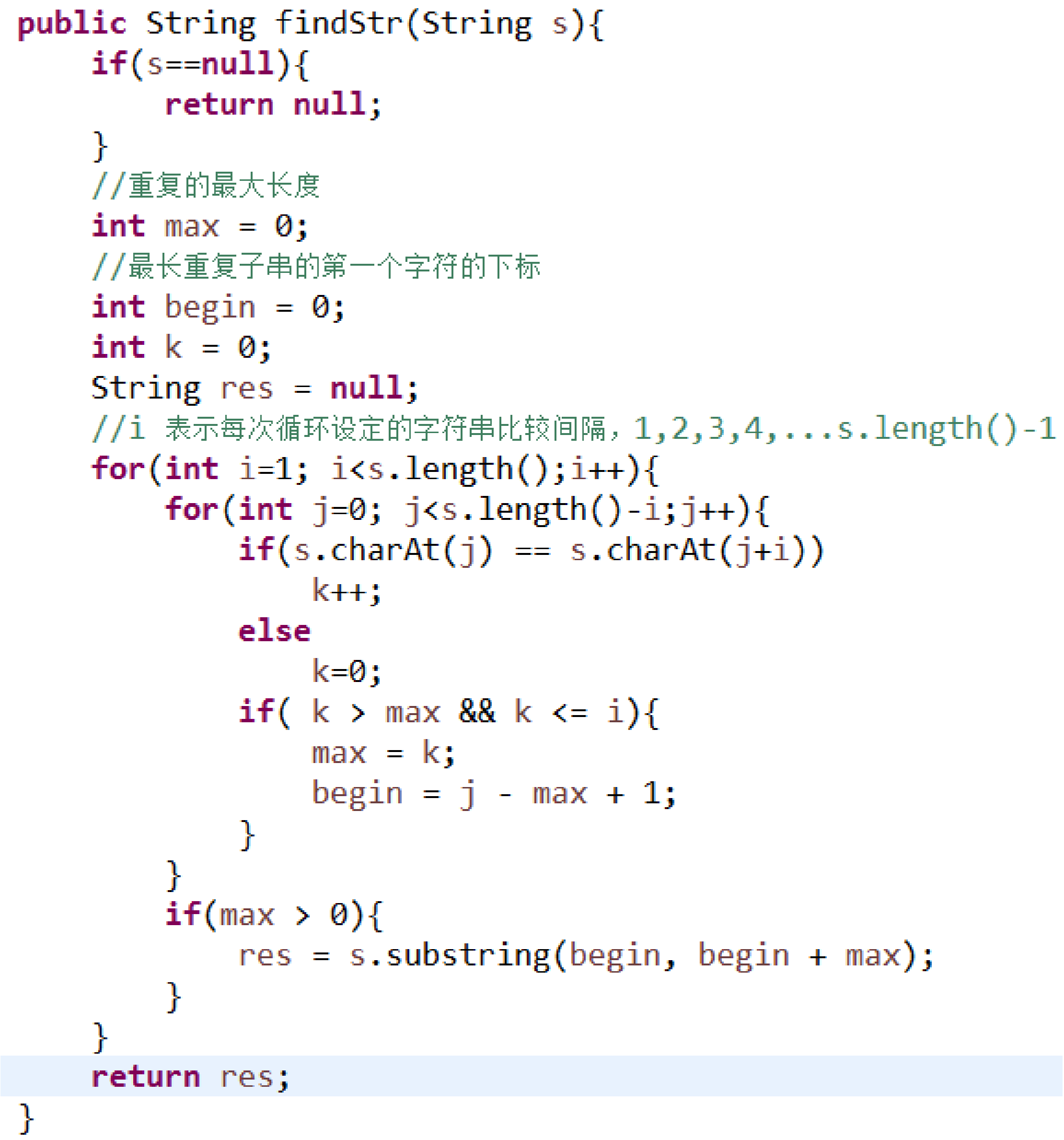
```python
def maxCommomSequnce(str1, str2):
    m = len(str1)
    n = len(str2)
    result = [[0]*n for i in range(m)]
    mylist = ""
    for i in range(m):
        for j in range(n):
            if str1[i] == str2[j]:
                result[i][j] = result[i-1][j-1] + 1
                mylist += str(str1[i]) + ' '
            else:
                result[i][j] = max(result[i-1][j], result[i][j])
    print mylist
    return result[m-1][n-1]
```

10、最长公共子序列



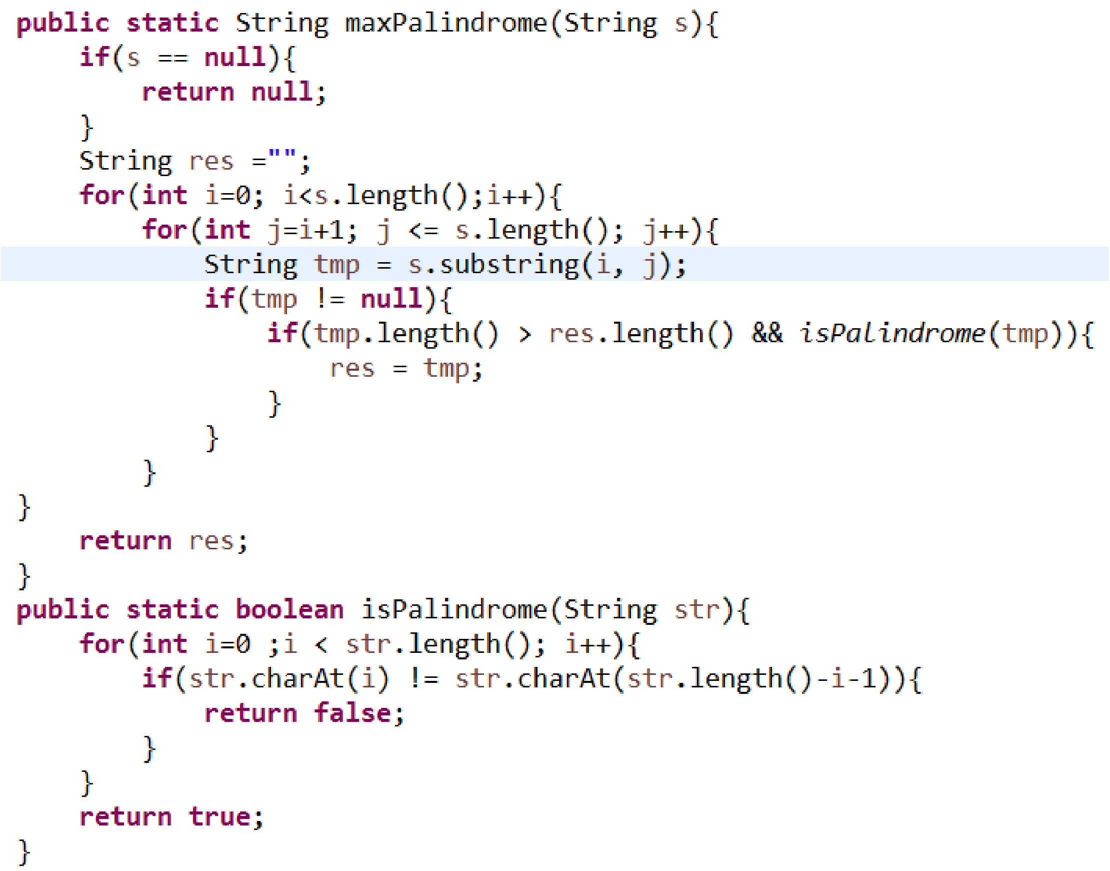
```python
def find_LCS(s1, s2):
    dp = [[0 for i in range(len(s2)+1)] for j in range(len(s1)+1)]
    d = [[None for i in range(len(s2)+1)] for j in range(len(s1)+1)]
    for i in range(len(s1)):
        for j in range(len(s2)):
            if s1[i] == s2[j] :
                dp[i+1][j+1] = dp[i][j] + 1
                d[i+1][j+1] = "ok"

            elif dp[i+1][j] > dp[i][j+1]:
                dp[i+1][j+1] = dp[i+1][j]
                d[i+1][j+1] = "left"
            else:
                dp[i+1][j+1] = dp[i][j+1]
                d[i+1][j+1] = "up"
    p1, p2 = len(s1), len(s2)
    result = []
    while d[p1][p2]:
        c = d[p1][p2]
        if c == 'ok':
            result.append(s1[p1-1])
            p1 -= 1
            p2 -= 1
        if c =='left':
            p2 -= 1
        if c == 'up':
            p1 -= 1
    result.reverse()
    return "".join(result)
```

11、最长不重复子串



```java
public String findStr(String s){
    if(s==null){
        return null;
    }
    //重复的最大长度
    int max = 0;
    //最长重复子串的第一个字符的下标
    int begin = 0;
    int k = 0;
    String res = null;
    //i 表示每次循环设定的字符串比较间隔，1,2,3,4,...s.length()-1
    for(int i=1; i<s.length();i++){
        for(int j=0; j<s.length()-i;j++){
            if(s.charAt(j) == s.charAt(j+i))
                k++;
            else
                k=0;
            if( k > max && k <= i){
                max = k;
                begin = j - max + 1;
            }
        }
        if(max > 0){
            res = s.substring(begin, begin + max);
        }
    }
    return res;
}
```

12、最长回文子串

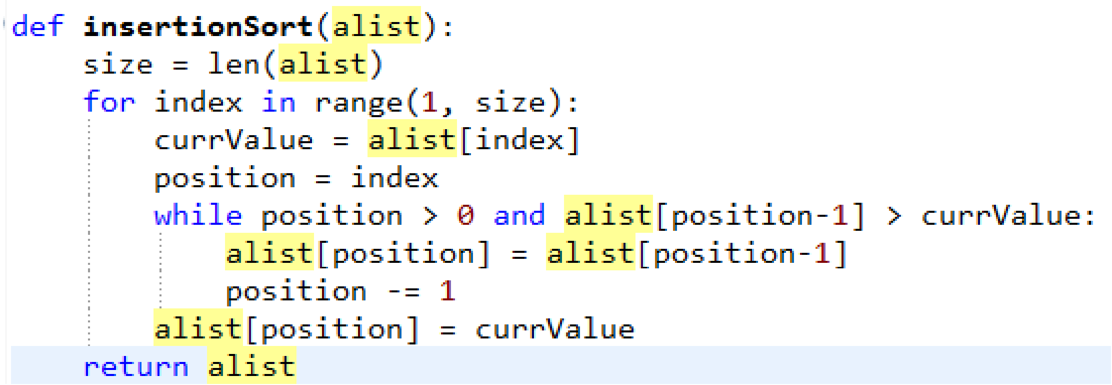


```java
public static String maxPalindrome(String s){
    if(s == null){
        return null;
    }
    String res ="";
    for(int i=0; i<s.length();i++){
        for(int j=i+1; j <= s.length(); j++){
            String tmp = s.substring(i, j);
            if(tmp != null){
                if(tmp.length() > res.length() && isPalindrome(tmp)){
                    res = tmp;
                }
            }
        }
}
    return res;
}
public static boolean isPalindrome(String str){
    for(int i=0 ;i < str.length(); i++){
        if(str.charAt(i) != str.charAt(str.length()-i-1)){
            return false;
        }
    }
    return true;
}
```

数据结构

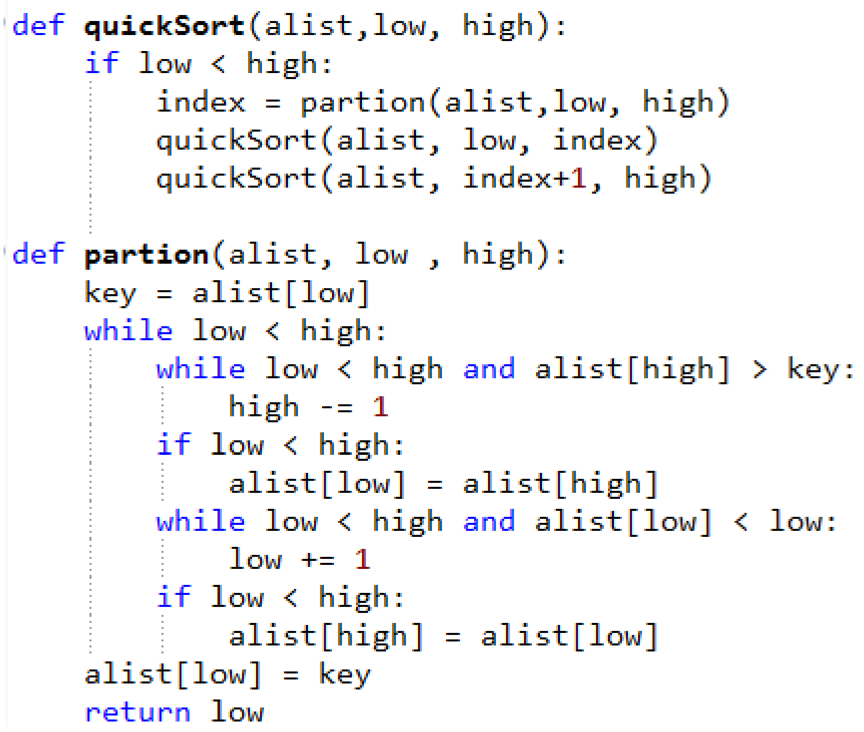
1、插入排序

选择序列中的第一个元素作为有序序列，逐渐将后面的元素插入到前面的有序序列中。



```python
def insertionSort(alist):
    size = len(alist)
    for index in range(1, size):
        currValue = alist[index]
        position = index
        while position > 0 and alist[position-1] > currValue:
            alist[position] = alist[position-1]
            position -= 1
        alist[position] = currValue
    return alist
```

2、快速排序



```
def quickSort(alist,low, high):
    if low < high:
        index = partion(alist,low, high)
        quickSort(alist, low, index)
        quickSort(alist, index+1, high)

def partion(alist, low , high):
    key = alist[low]
    while low < high:
        while low < high and alist[high] > key:
            high -= 1
        if low < high:
            alist[low] = alist[high]
        while low < high and alist[low] < low:
            low += 1
        if low < high:
            alist[high] = alist[low]
    alist[low] = key
    return low
```

3、