两数之和

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        hashmap = {}
        for index, num in enumerate(nums):
            if hashmap.get(target-num) is not None:
                return [hashmap.get(target-num),index]
            hashmap[num] = index
```

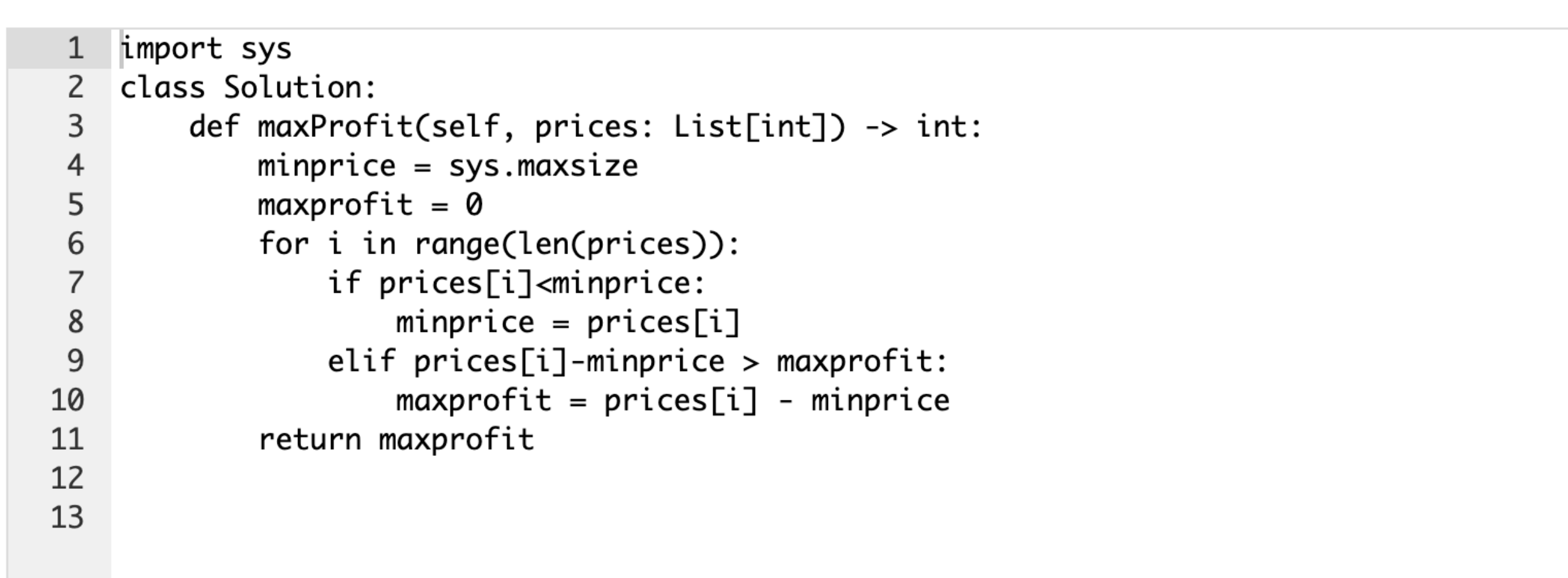
两个有序列表的交集

```
class Solution:
    def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
        set1 = set(nums1)
        set2 = set(nums2)
        return list(set2 & set1)
```

两个有序列表的中位数

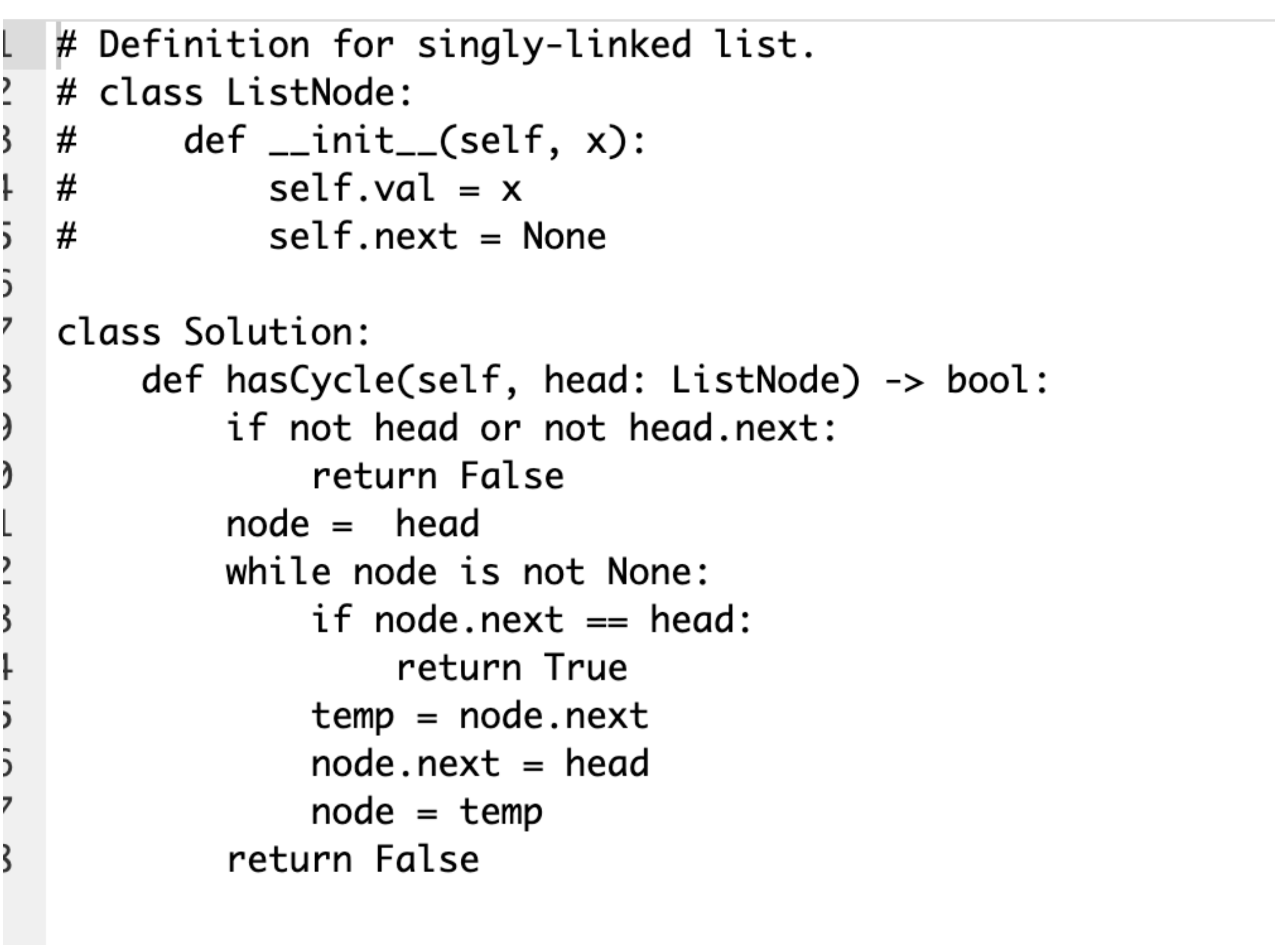
```
class Solution:
    def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) -> float:
        m, n = len(nums1), len(nums2)
        if m > n:
            nums1, m, nums2, n = nums2, n, nums1, m
        l, r, half_pos = 0, m, (m + n + 1) // 2
        while l <= r:
            i = l + (r - l) // 2
            j = half_pos - i
            if i > 0 and nums1[i-1] > nums2[j]:
                r = i - 1
            elif i < m and nums1[i] < nums2[j-1]:
                l = i + 1
            else:
                if i == 0:
                    max_of_left = nums2[j-1]
                elif j == 0:
                    max_of_left = nums1[i-1]
                else:
                    max_of_left = max(nums1[i-1], nums2[j-1])
                if (m + n) % 2 == 1:
                    return max_of_left
                if i == m:
                    min_of_right = nums2[j]
                elif j == n:
                    min_of_right = nums1[i]
                else:
                    min_of_right = min(nums1[i], nums2[j])
                return (max_of_left + min_of_right) / 2.0
```

股票买卖的最佳时机



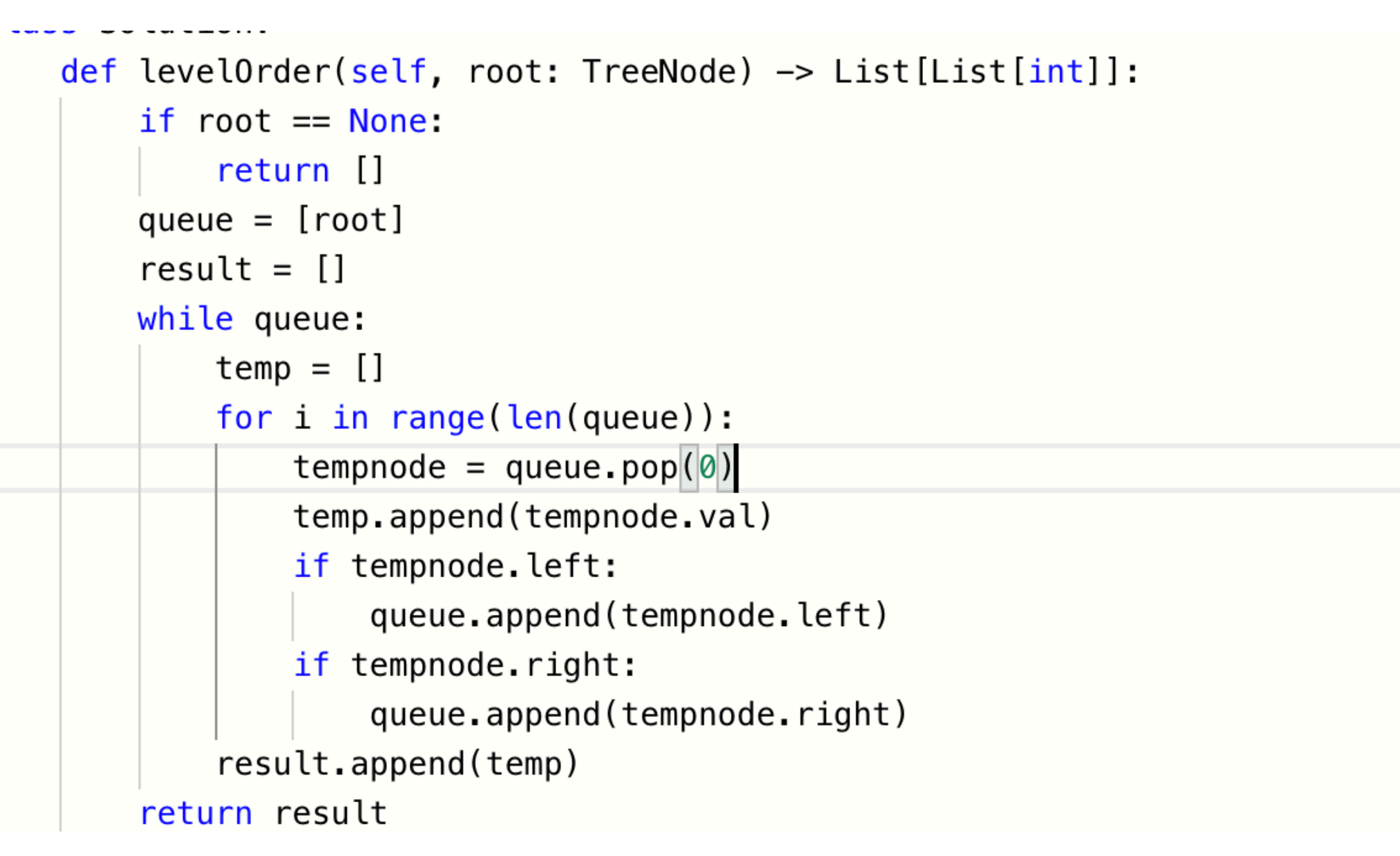
```
import sys
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        minprice = sys.maxsize
        maxprofit = 0
        for i in range(len(prices)):
            if prices[i]<minprice:
                minprice = prices[i]
            elif prices[i]-minprice > maxprofit:
                maxprofit = prices[i] - minprice
        return maxprofit
```

链表是否有环



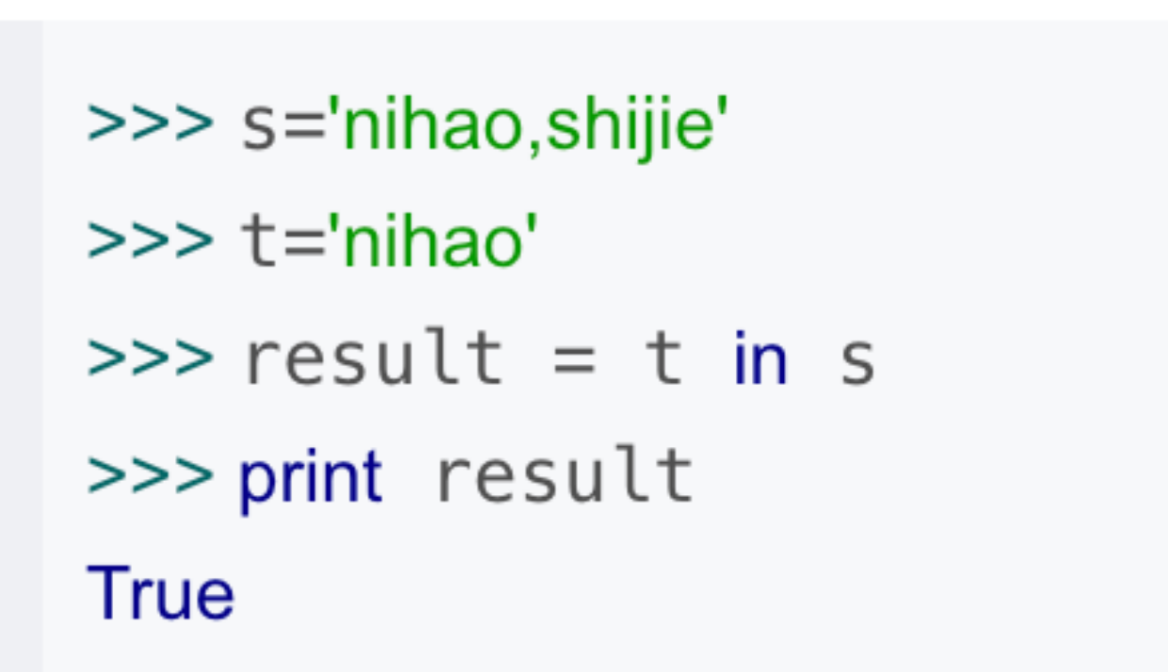
```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        if not head or not head.next:
            return False
        node =  head
        while node is not None:
            if node.next == head:
                return True
            temp = node.next
            node.next = head
            node = temp
        return False
```

二叉树层次遍历



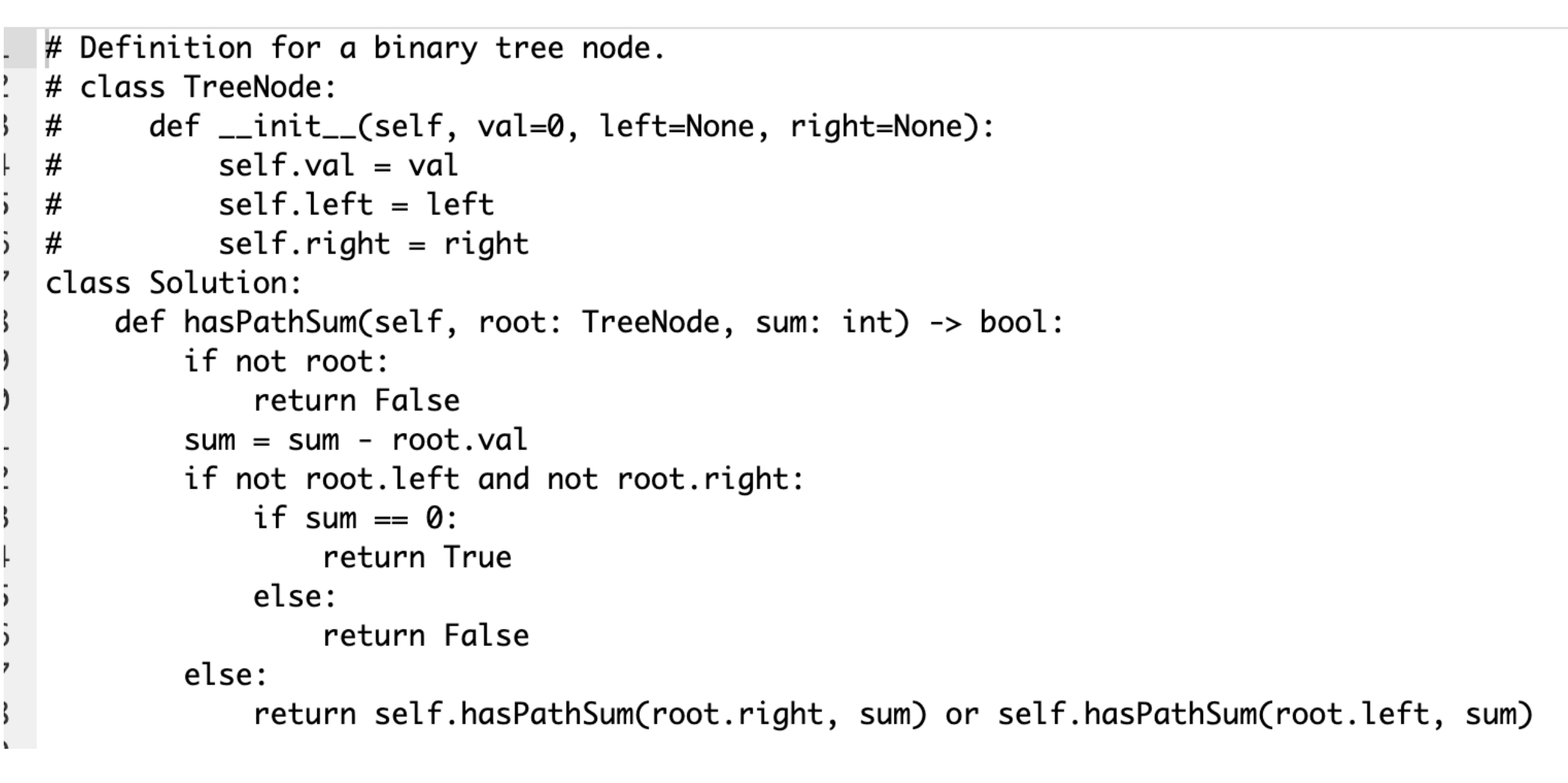
```
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if root == None:
            return []
        queue = [root]
        result = []
        while queue:
            temp = []
            for i in range(len(queue)):
                tempnode = queue.pop(0)
                temp.append(tempnode.val)
                if tempnode.left:
                    queue.append(tempnode.left)
                if tempnode.right:
                    queue.append(tempnode.right)
            result.append(temp)
        return result
```

两个字符串a和b，判断b是否为a的子串



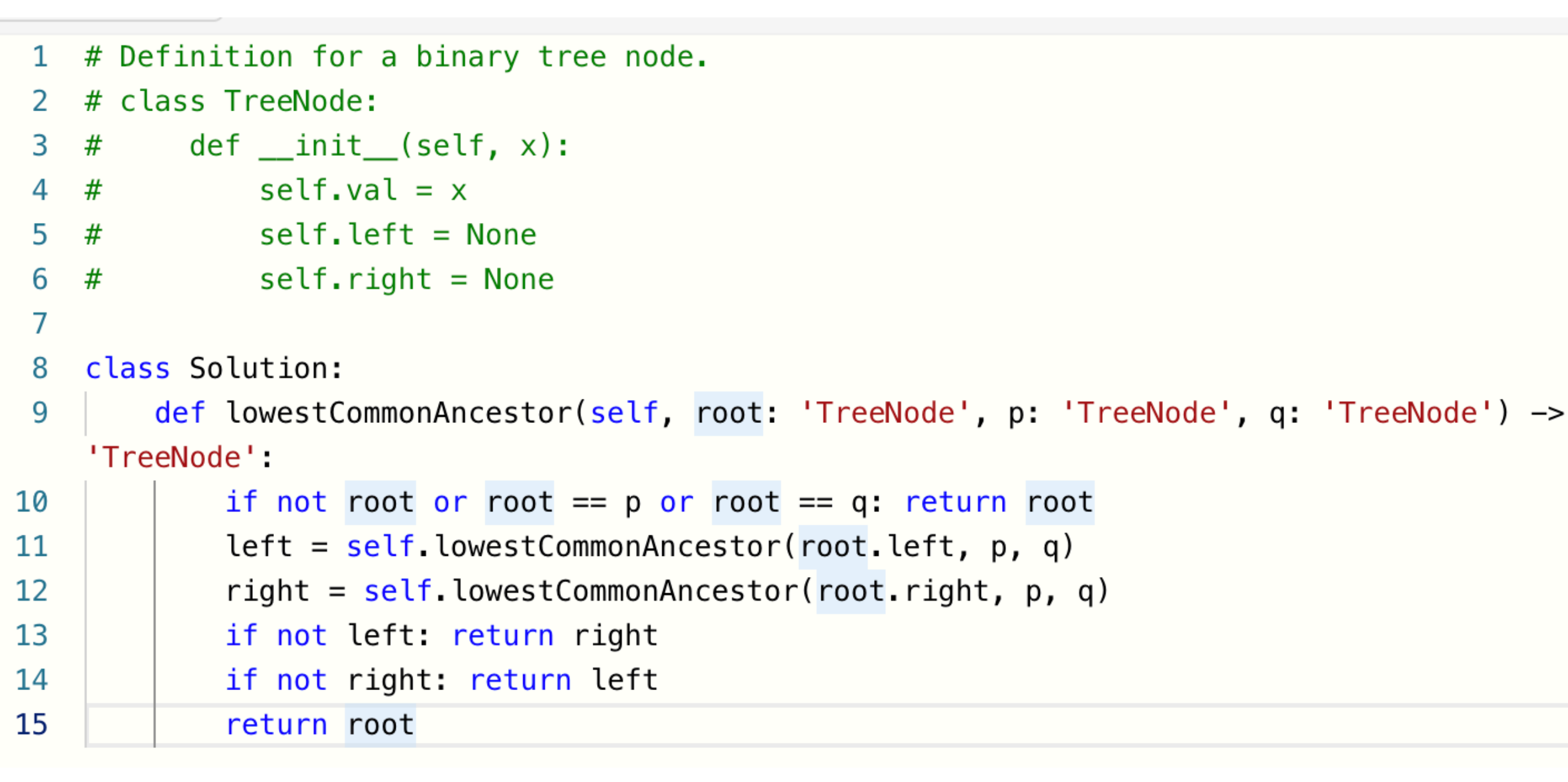
```
>>> s='nihao,shijie'
>>> t='nihao'
>>> result = t in s
>>> print result
True
```

二叉树路径之和



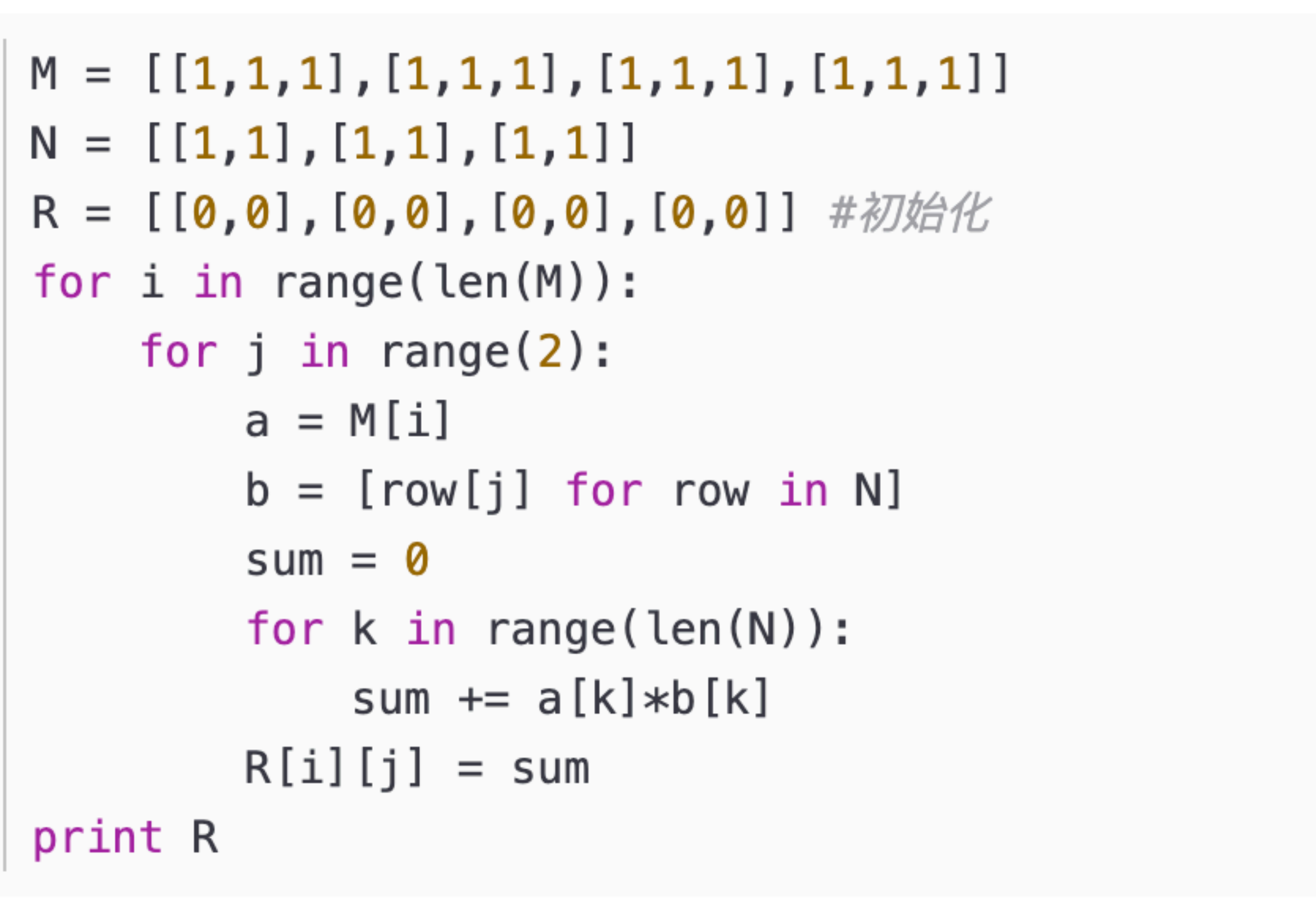
```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def hasPathSum(self, root: TreeNode, sum: int) -> bool:
        if not root:
            return False
        sum = sum - root.val
        if not root.left and not root.right:
            if sum == 0:
                return True
            else:
                return False
        else:
            return self.hasPathSum(root.right, sum) or self.hasPathSum(root.left, sum)
```

无序二叉树的公共祖先



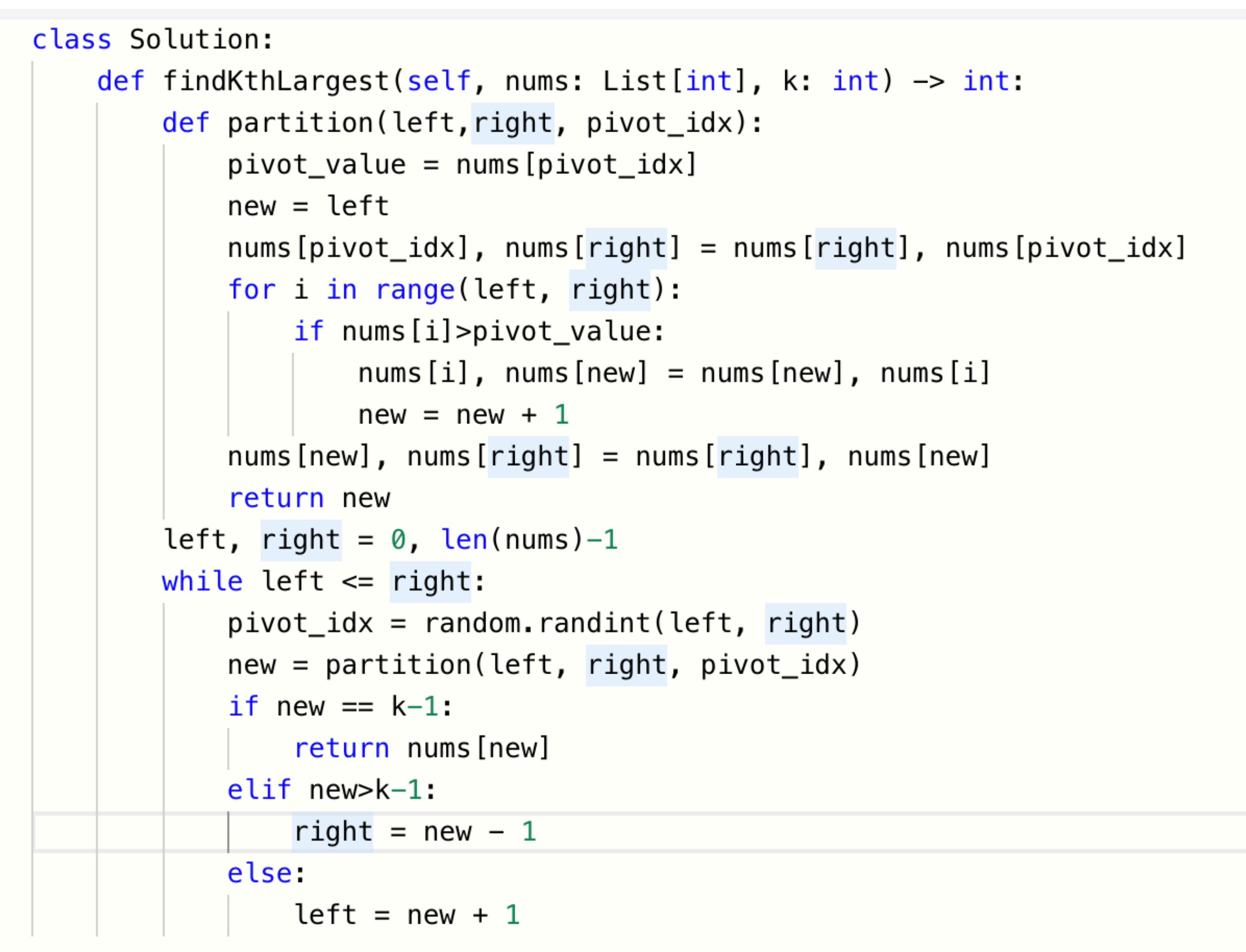
```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') ->
'TreeNode':
        if not root or root == p or root == q: return root
        left = self.lowestCommonAncestor(root.left, p, q)
        right = self.lowestCommonAncestor(root.right, p, q)
        if not left: return right
        if not right: return left
        return root
```

矩阵相乘



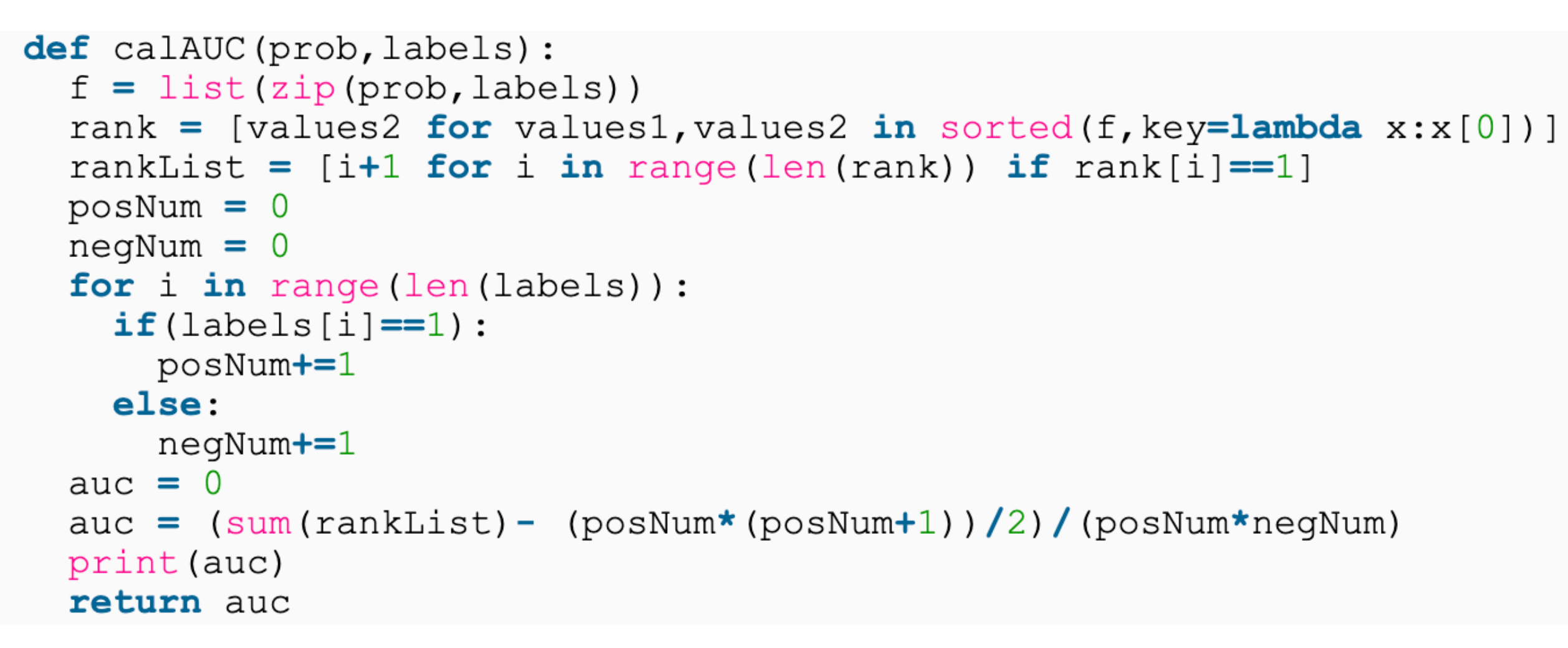
```
M = [[1,1,1],[1,1,1],[1,1,1],[1,1,1]]
N = [[1,1],[1,1],[1,1]]
R = [[0,0],[0,0],[0,0],[0,0]] #初始化
for i in range(len(M)):
    for j in range(2):
        a = M[i]
        b = [row[j] for row in N]
        sum = 0
        for k in range(len(N)):
            sum += a[k]*b[k]
        R[i][j] = sum
print R
```

寻找第k大的数



```
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        def partition(left,right, pivot_idx):
            pivot_value = nums[pivot_idx]
            new = left
            nums[pivot_idx], nums[right] = nums[right], nums[pivot_idx]
            for i in range(left, right):
                if nums[i]>pivot_value:
                    nums[i], nums[new] = nums[new], nums[i]
                    new = new + 1
            nums[new], nums[right] = nums[right], nums[new]
            return new
        left, right = 0, len(nums)-1
        while left <= right:
            pivot_idx = random.randint(left, right)
            new = partition(left, right, pivot_idx)
            if new == k-1:
                return nums[new]
            elif new>k-1:
                right = new - 1
            else:
                left = new + 1
```

徒手实现auc计算



```
def calAUC(prob,labels):
  f = list(zip(prob,labels))
  rank = [values2 for values1,values2 in sorted(f,key=lambda x:x[0])]
  rankList = [i+1 for i in range(len(rank)) if rank[i]==1]
  posNum = 0
  negNum = 0
  for i in range(len(labels)):
    if(labels[i]==1):
      posNum+=1
    else:
      negNum+=1
  auc = 0
  auc = (sum(rankList)- (posNum*(posNum+1))/2)/(posNum*negNum)
  print(auc)
  return auc
```

> 其中输入prob是得到的概率值，labels是分类的标签（1，-1）

损失函数有哪些以及定义?

https://zhuanlan.zhihu.com/p/58883095

- 损失函数用来评价模型的预测值和真实值的不一致程度。
  损失函数分为经验风险损失函数和结构风险损失函数。
- 0-1损失函数[直接对应分类判断错误的个数，但是它是一个非凸函数,不太适用] ----感知机

$$L(Y, f(X)) = \begin{cases} 1, |Y - f(X)| \geq T \\ 0, |Y = f(X)| < T \end{cases}$$

- 绝对值损失函数

$$L(Y, f(x)) = |Y - f(x)|$$

- 平方损失函数 ---应用于回归问题
- log对数损失函数 ----逻辑回归

$$L(Y, P(Y|X)) = -logP(Y|X)$$

  （1）log对数损失函数能非常好的表征概率分布，在很多场景尤其是多分类，如果需要知道结果属于每个类别的置信度，那它非常适合。
  （2）健壮性不强，相比于hinge loss对噪声更敏感。
  （3）逻辑回归的损失函数就是log对数损失函数
- hinge损失函数 ---svm

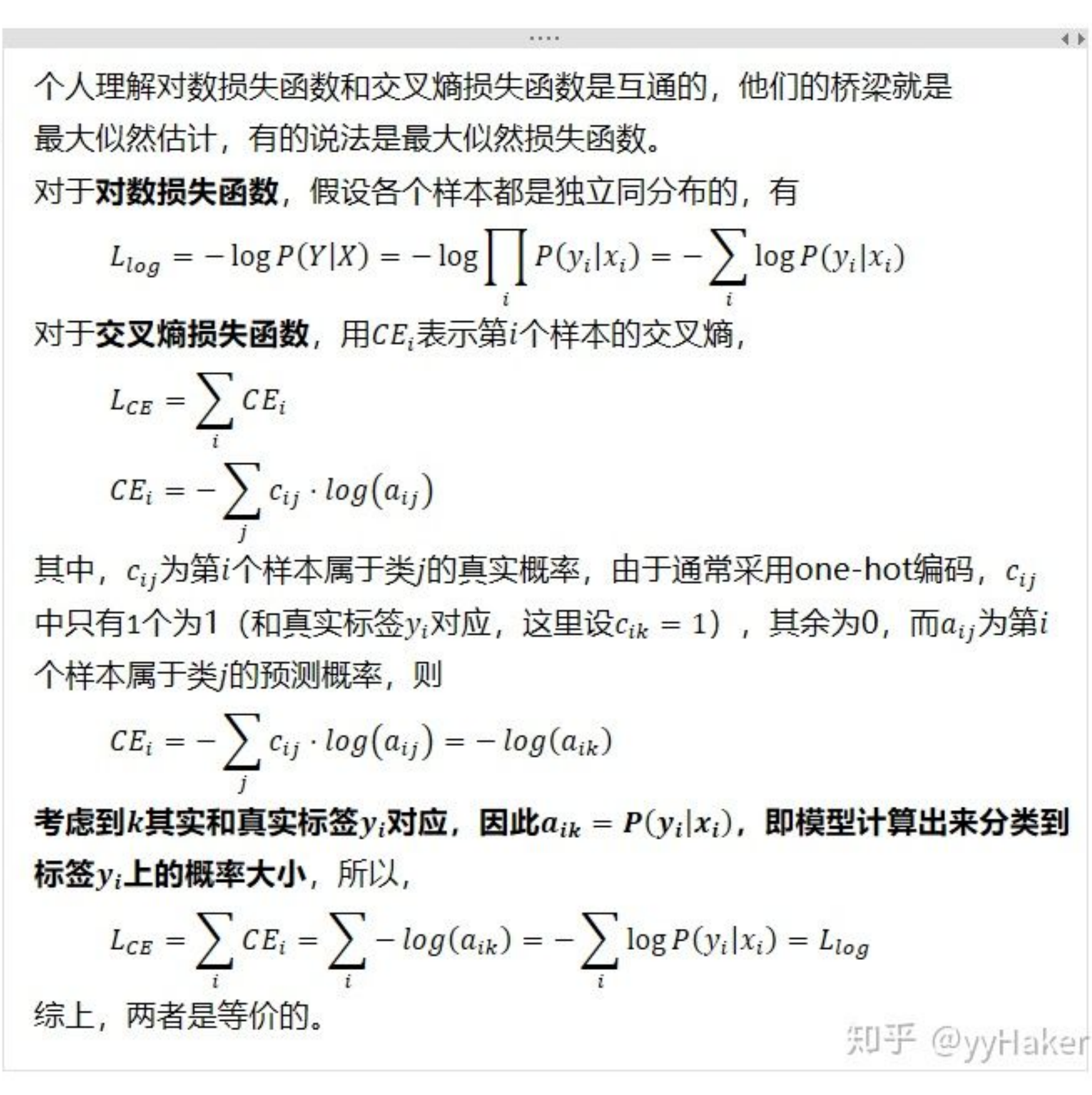
$$L(y, f(x)) = max(0, 1 - yf(x))$$

- 感知损失(perceptron loss)函数

$$L(y, f(x)) = max(0, -f(x))$$

- 交叉熵损失函数 ---二分类或者多分类任务中 softmax或者sigmoid后接

$$C = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln(1 - a)]$$



个人理解对数损失函数和交叉熵损失函数是互通的，他们的桥梁就是最大似然估计，有的说法是最大似然损失函数。

对于**对数损失函数**，假设各个样本都是独立同分布的，有

$$L_{log} = -\log P(Y|X) = -\log \prod_i P(y_i|x_i) = -\sum_i \log P(y_i|x_i)$$

对于**交叉熵损失函数**，用CE_i表示第i个样本的交叉熵，

$$L_{CE} = \sum_i CE_i$$

$$CE_i = -\sum_j c_{ij} \cdot log(a_{ij})$$

其中，c_{ij}为第i个样本属于类j的真实概率，由于通常采用one-hot编码，c_{ij}中只有1个为1（和真实标签y_i对应，这里设$c_{ik} = 1$），其余为0，而a_{ij}为第i个样本属于类j的预测概率，则

$$CE_i = -\sum_j c_{ij} \cdot log(a_{ij}) = -log(a_{ik})$$

考虑到k其实和真实标签y_i对应，因此$a_{ik} = P(y_i|x_i)$，即模型计算出来分类到标签y_i上的概率大小，所以，

$$L_{CE} = \sum_i CE_i = \sum_i -log(a_{ik}) = -\sum_i \log P(y_i|x_i) = L_{log}$$

综上，两者是等价的。

数据进行归一化的方法有哪些？为什么要进行归一化，哪些模型一定需要归一化

https://zhuanlan.zhihu.com/p/87610305
https://www.zhihu.com/question/20455227

- 为什么
  我们需要将这些量纲不同的属性数据映射到同一个尺度空间中，这就是归一化
  （1）归一化后加快了梯度下降求最优解的速度
  （2）归一化有可能提高精度
  一些分类器需要计算样本之间的距离（如欧式距离，knn），如果一个特征值域范围非常大，那么距离计算就主要取决于这个特征，从而与实际情况相悖（比如这时实际情况是值域范围小的特征更重要）。
- 归一化的方式
  （1）最值归一化(normalization)

  定义：把所有数据映射到 0-1 之间。

  计算公式：$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$

  适用情况：适用于分布有明显边界的情况，但受 outlier 值的影响比较大。这个可以从公式的分母这一项进行理解，如果 x_{max} 和 x_{min} 相差很大并且整个数据样本中 x_{max} 和 x_{min} 这种极端样本比较少，那你会发现数据分布还是不均衡，接近于 1 的值在值为 x_{max} 这些特征中，接近于 0 的值在值为 x_{min} 的这些特征中。

  （2）均值方差归一化(standardization)

  定义：把同一特征的数据归一化到均值为 0 方差为 1 的分布中。

  计算公式：$x_{scaled} = \frac{x - x_{mean}}{S}$ (s 代表某一特征的方差)

  适用情况：数据分布没有明显的边界；有可能存在极端数据值。通常这种归一化方式适用于所有情况。
- 哪些模型需要归一化
  概率模型不需要归一化，因为它们不关心变量的值，而是关心变量的分布和变量之间的条件概率，如决策树，rf。而像adaboost、gbdt、xgboost、svm、lr、KNN、KMeans之类的最优化问题就需要归一化。