# DS Python Practice Questions

> Miranda Zhao

**1. Python Kick Off**

- Hello World, Pound to Kilogram, Odd Number, FizzBuzz, Square Root

**2. List I**

- List Comprehension, OddNums, C2F, Fibonacci List, Reverse List
- Whether an Element in the List, Remove an Element, Separate Numbers, Remove Duplicates from a List

**3. List II**

- Sort a List, Insert a Number in a Sorted List, Remove duplicates from a sorted list
- Merge Two Sorted Lists, List to Number, Maximum Stock Gain, Sort List of Lists, Pascals Triangle

**4. String**

- All Red, Reverse String, Reverse Sentence, Reverse Words
- Capitalize First Letters, Remove Punctuations, N Grams
- Palindrome, Letter Palindrome, Valid Parenthesis

**5. Dictionary**

- Key Value Switch, Rank Keys by Values, Character Frequency
- Unique Only, Same Pattern, Anagrams, Target Sum

# 1.Python Kick Off

## 1.1 Hello World

Description Build a function HelloWorld which returns "Hello World".

Examples HelloWorld( ) return "Hello World"

In [1]:
```
def HelloWorld():
    return "Hello World"
```

## 1.2 Pound to Kilogram

Description Build a function LB2KG which takes a value in pound as input and returns the corresponding value in kilogram. One pound equals to 0.4536 kg.

Examples LB2KG(1) returns 0.4536

In [2]:
```
def LB2KG(pound):
    kilogram = 0.4536 * pound
    return kilogram
```

## 1.3 Odd Number

Description Build a function Odd which takes an integer as input and returns a boolin to tell whether the integer is odd or not.

Examples Odd(3) returns True

Odd(4) returns False

In [3]:
```
#if后和else后都有:
def Odd(num):
    if num % 2 != 0:
        return True
    else:
        return False
```

## 1.4 FizzBuzz

Description Build a function FizzBuzz which takes a positive integer as input. If the number is divisible by 3, the function returns "Fizz". If the number is divisible by 5, the function returns "Buzz". If the number is divisible by both 3 and 5, the function returns "FizzBuzz". Otherwise, the function returns the number itself.

Examples FizzBuzz(45) returns "FizzBuzz"

FizzBuzz(9) returns "Fizz"

FizzBuzz(25) returns "Buzz"

FizzBuzz(17) returns "17"

In [4]:
```
def FizzBuzz(num):
    if num % 3 == 0 and num % 5 == 0:
        return "FizzBuzz"
    if num % 3 == 0:
        return "Fizz"
    if num % 5 == 0:
        return "Buzz"
    else:
        return num

#① % 代表整除取余
#② == 代表判断，不可以用=代替
#② 在逻辑上需要最先判断除以3和除以5均无余数的情况
#③ 有3个if 1个else；有1个if 2个elif 1个else 都可以
```

## 1.5 Square Root

Description Build a function sqrt to calculate the square root of a positive integer number. If the result is not a integer, the function returns the nearest integer to the left of the result number.

Examples sqrt(16) returns 4

sqrt(17) returns 4

sqrt(77) returns 8

sqrt(1) returns 1

In [5]:
```
import math

def sqrt(num):
    result = math.sqrt(num)
    return math.floor(result)

sqrt(16) #4
sqrt(17) #4
```

Out[5]: 4

In [6]:
```
def sqrt(num):
    if num < 0:
        raise ValueError("Input must be a non-negative integer.")
    if num == 0:
        return 0

    for i in range(1, num + 1):
        if i * i > num:
            return i - 1

    return num

sqrt(16) #4
sqrt(17) #4

#https://pythontutor.com/render.html#code=def%20sqrt%28num%29%3A%0A%2
#https://pythontutor.com/render.html#code=def%20sqrt%28num%29%3A%0A%2
```

Out[6]: 4

In [7]:
```
def sqrt(num):
    if num < 0:
        raise ValueError("Input must be a non-negative integer.")
    if num == 0:
        return 0

    left = 1
    right = num

    while left <= right:
        mid = (left + right) // 2
        square = mid * mid

        if square == num:
            return mid
        elif square < num:
            left = mid + 1
        else:
            right = mid - 1

    return right
```

# 2. List I

## 2.2 List Comprehension

Description Build a function LstComp which takes two integers (i and j) as inputs and returns a list of numbers from i to j.

Examples LstComp(1, 5) returns [1, 2, 3, 4, 5]

合并一个值域进list

In [8]:
```
#法1:
def LstComp(i, j):
    a = []
    for k in range(i, j+1):
        a.append(k)
    return a
```

In [9]:
```
#法2:
def LstComp(i, j):
    return [k for k in range(i, j+1)]
```

## 2.2 OddNums

Description Build a function OddNums which takes two integers (i and j) as inputs and returns a list of ODD numbers between i and j inclusively.

Examples OddNums(1, 5) returns [1, 3, 5]

OddNums(2, 6) returns [3, 5]

OddNums(1, 2) returns [1] 返回奇数

In [10]:
```
#法1：
#def后；for后；if后 都有：
def OddNums(i, j):
    a = []
    for k in range(i, j+1):
        if k % 2 != 0:                #k是奇数
            a.append(k)
    return a

#注意：
#①：先建立一个空集a，如果是奇数，append到a中，最后return a
#②：if k % 2 !=0 后面要加：
#③：append的写法：list.append(num)
```

In [11]:
```
#法2：return [k for k in range(a, b) if xxx]
def OddNums(i, j):
    return [k for k in range(i, j+1) if k % 2 !=0]
```

## 2.3 C2F

Description Build a function C2F which converts a list of temperatures in degrees from Celsius to Fahrenheit. F=C*1.8+32.

Examples C2F([0, 38]) returns [32.0, 100.4]

In [12]:
```
#①[ax + b for C in Cs]写法：
def C2F(Cs):
    return [C * 1.8 + 32 for C in Cs]

#为什么必须加[]：
#在给定的代码中，方括号[]用于创建列表推导式（list comprehension）。
#列表推导式是一种简洁的方式，通过迭代现有可迭代对象（在这种情况下是列表Cs）并对每
#[]是一个列表推导式，它通过迭代输入列表Cs中的每个元素C*1.8+32，并将结果添加到新
```

In [13]:
```
#②用for loop写法：
#result为空集，for C in Cs时，F = ax + b，把每个F append进result中，最后re
def C2F(Cs):
    result = []
    for C in Cs:
        F = C * 1.8 + 32
        result.append(F)
    return result
```

## 2.4 Fibonacci List

Description The Fibonacci List is the list of numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, … The next number is generated by adding up the two numbers before it. Build a function FibLst which takes an integer n as input and returns an n-element Fibonacci List. If n is less than 1, returns -1.

Examples FibLst(1) returns [0]

FibLst(3) returns [0, 1, 1]

FibLst(6) returns [0, 1, 1, 2, 3, 5]

FibLst(-3) returns -1

斐波纳契数列：后一个数是前两个数之和

In [14]:
```
def Fibonacci(n):
    if n < 1:
        return -1
    a = []
    for i in range(n):
        if i == 0:
            a.append(0)
        elif i == 1:
            a.append(1)
        else:
            a.append(a[i - 1] + a[i - 2])
    return a

#①：建立空list a，对于所有range(n)内的值i，如果是0，则赋值0，如果是1，则赋值1
#②：if n < 1的edge case，和for loop是对齐的。
#③：for loop下的if, elif, else分类讨论，else后不需要再写n>1等解释；elif不能
#④：判断用==，不能写成=
#⑤将每个数房间list用list.append()。如果加0/1直接赋值，就在括弧里写0/1。如果加
#⑥return与for loop进行对齐

Fibonacci(1)
#i in range(1) i = 0      #[0]
Fibonacci(2)
#i in range(2) i = 0,1    #[0, 1]
Fibonacci(3)
#i in range(3) i = 0,1,2  a[1]+a[0]=1 #[0, 1, 1]
Fibonacci(4)
#i in range(4) i = 0,1,2,3 a[1]+a[0]=1 a[2]+a[1]=2 #[0, 1, 1, 2]
```

Out[14]:
```
[0, 1, 1, 2]
```

## 2.5 Reverse List

Description Build a function RevLst which reverses a list.

Examples RevLst([3, 1, 9, 8, 4]) returns [4, 8, 9, 1, 3]

反转list

In [15]:
```
#法0：左右手，左手定为第一位置，右手是最末位置；当左小于右，让两者调换
#左+1，右-1；再两者调换，直到右小于左
def RevLst(nums):
    left = 0
    right = len(nums) - 1
    while left < right:
        nums[left], nums[right] = nums[right], nums[left]
        left += 1
        right -= 1
    return nums
```

In [16]:
```
#方法1：list.reverse()
def RevLst(nums):
    nums.reverse()
    return nums

#注意：先写一行list.reverse()，再写return list，两者对齐
#注意要写两行，不能直接return nums.reverse()。
```

In [17]:
```
#为什么不能写成：
# def RevLst(a):
#     return a.reverse()

#因为a.reverse()是一个动作，直接return返回的是None。需要操作动作后，返回a。
#a.reverse() 方法在原地反转列表 a，并且不返回任何值，即它返回的是 None。所以实
#先使用 a.reverse() 来原地修改列表 a，然后再使用 return a 返回反转后的列表。这
#reverse() 方法是一个原地操作，它会修改列表本身，并不返回任何值。因此需要在 ret
```

In [18]:
```
#法2：list[::-1]
def RevLst(nums):
    return nums[::-1]

#注意：只用写一行return list[::-1]
```

In [19]:
```
#法3：
def RevLst(nums):
    left = 0
    right = len(nums) - 1
    while left < right:
        nums[left], nums[right] = nums[right], nums[left]
        left += 1
        right -= 1
    return nums
```

## 2.6 Whether an Element in the List

Description Build a function IsIn to tell whether a target element in a list.

Examples IsIn([1, 2, 3, 4, 5], 2) returns True IsIn([1, 2, 3, 4, 5], 6) returns False

是否元素在list中的函数形式

In [20]:
```
#法1：if k in a:
def IsIn(nums, target):
    if target in nums:
        return True
    else:
        return False

#一个值在一个列表中：if a in list, return True False
```

In [21]:
```
#法2：if target in nums:
def IsIn(nums, target):
    return target in nums

#直接return k in a, 返回结果是True False
#return target in nums 是一个布尔表达式，它会返回一个布尔值，即 True 或 Fals
```

## 2.7 Remove an Element

Description Build a function RmEl to remove a target element from a list. If the target element appears in the list for multiple times, only remove the first one.

Examples RmEl([1, 2, 3, 4, 5], 2) returns [1, 3, 4, 5]

RmEl([1, 2, 3, 4, 5], 6) returns [1, 2, 3, 4, 5]

RmEl([1, 2, 3, 3, 4, 5], 3) returns [1, 2, 3, 4, 5]

In [22]:
```
def RmEl(nums, target):
    if target in nums:
        nums.remove(target)
    return nums

#①：要检查target在nums中才可以进行remove操作
#②：if k in a: 直接操作a.remove(k)，再return a。不用写nums = XXX。
#③：return 应与if对齐
```

## 2.8 Separate Numbers

Description Build a function SepNum which extracts the odd numbers from a given list ands append the odd numbers at the end of the list.

Examples SepNum([1, 2, 3, 4, 5, 6]) returns [2, 4, 6, 1, 3, 5]

SepNum([1, 3, 5]) returns [1, 3, 5]

SepNum([2, 4, 6]) returns [2, 4, 6]

In [23]:
```
def SepNum(nums):
    odds = []
    evens = []
    for num in nums:
        if num % 2 == 0:            #放偶数
            evens.append(num)
        else:                       #放奇数
            odds.append(num)
    return evens + odds

#①：分别建立偶数空集和奇数空集，对于在nums中的num，如果除以2余数为0，append到
#②：for数字在列表中，可以写for num in nums。
#③：两个list相连直接相加即可
#④：在if 和else后分别进行append操作，return两个list相连与 for loop对齐。
```

## 2.9 Remove Duplicates from a List

Description Build a function RmDuplicate which remove all the duplicates from a list.

Examples RmDuplicate([3, 1, 2, 3, 4, 5, 6]) returns [1, 2, 3, 4, 5, 6]

RmDuplicate([1, 3, 5]) returns [1, 3, 5]

移除重复元素并返回更新后列表。

In [24]:
```
#法1：如果num没在a空集中，就把num append进a中，并进行sort
def RmDuplicate(nums):
    a = []
    for num in nums:
        if num not in a:
            a.append(num)
            a.sort()
    return a

#①：只需要建立一个最后需要return的a集，
#②：for num in nums, if num not in a, a.append(num) 可以写成步步缩进格式
#即在num在nums中的时候，如果num没在a里，就把其放进去。
#③：在append后，要进行sort。
#④：return 和 for进行对齐。
```

# 3. List II

## 3.1 Sort a List

Description Build a function SortLst which sorts a list. At this time you can just use the built-in function in Python.

Examples SortLst([4, 2, 1, 3, 4, 5]) returns [1, 2, 3, 4, 4, 5]

SortLst([1, 3, 5]) returns [1, 3, 5]

给序列排序

In [25]:
```
#写法1：list.sort(); return nums
def SortLst(nums):
    nums.sort()
    return nums
```

In [26]:
```
#写法2：return sorted(nums)
def SortLst(nums):
    return sorted(nums)
```

In [27]:
```
#写法3：list = sorted(list) return list
def SortLst(nums):
    nums = sorted(nums)
    return nums
```

## 3.2 Insert a Number in a Sorted List

Description Build a function NumInsert which inserts a number in a sorted list.

Examples NumInsert([1, 2, 3, 4, 5, 6], 3) returns [1, 2, 3, 3, 4, 5, 6]

NumInsert([1, 2, 3, 4, 5, 6], 7) returns [1, 2, 3, 4, 5, 6, 7]

NumInsert([1, 2, 3, 4, 5, 6], -1) returns [-1, 1, 2, 3, 4, 5, 6]

在list中插入一个值

In [28]:
```
#写法0：遍历法
#让ind = list长度，从头至尾遍历所有元素，当list中的元素大于插入值时，让ind = i
#list.insert(index, target)，让index位置插入target元素
def NumInsert(nums, target):
    ind = len(nums)
    for i in range(len(nums)):
        if nums[i] >= target:
            ind = i
            break
    nums.insert(ind, target)
    return nums
```

In [29]:
```
#写法1: 二分法
#步骤:
# 1.设定左右两个指针，分解为L和R
# 2.L初始值为0，R初始值为List长度
# 3.计算中间指针的值 M = (left + right) / 2
#     3.1. 若M指针对应的数，大于需要插入的数，将M设为R
#     3.2. 若M指针对应的数，小于需要插入的数，将M设为L
# 4.重复步骤3，直到
#     4.1M指针对应的值等于需要插入的值
#     4.2或者，左右指针相遇
# 5.将目标数字插入相应的位置

#每次取一半，把另一半扔掉，比从头到尾遍历要快

def NumInsert(nums, target):
    left = 0
    right = len(nums)
    while right - left > 1:
        mid = (left + right) // 2    #整数除法 //: 向下取整
        if nums[mid] == target:
            nums.insert(mid, target)
            return nums
        elif nums[mid] > target:
            right = mid
        else:
            left = mid
    if target > nums[mid]:
        nums.insert(right, target)
    else:
        nums.insert(left, target)
    return nums
```

In [30]:
```
#写法3: return sorted(nums)
def NumInsert(nums, target):
    nums.append(target)
    return sorted(nums)
```

In [31]:
```
#写法4: nums = sorted(nums) return nums
def NumInsert(nums, target):
    nums.append(target)
    nums = sorted(nums)
    return nums
```

In [32]:
```
#写法4: nums.sort() return nums
def NumInsert(nums, target):
    nums.append(target)
    nums.sort()
    return nums
```

In [33]:
```
#写法5:
def NumInsert(lst, num):
    return sorted(lst + [num])
```

In [34]:
```
#写法6：在 i 小于 list 长度时，且 lst[i] < 插入数 时，lst[i]向后移动，
#直到lst[i] >= 插入值时停止，返回i前原数列 + 插入值 + i后原数列
def NumInsert(lst, num):
    i = 0
    while i < len(lst) and lst[i] < num:
        i += 1
    return lst[:i] + [num] + lst[i:]
```

## 3.3 Remove duplicates from a sorted list

Description Build a function SortRm which removes duplicate numbers in a sorted list.

Examples SortRm([1, 2, 2, 3, 4, 4, 5, 6]) returns [1, 2, 3, 4, 5, 6]

SortRm([1, 2, 3, 4, 5, 6]) returns [1, 2, 3, 4, 5, 6]

从排好序的列表中移除重复值

In [35]:
```
#写法1：正向：如果target【在】原list中，【remove】
def RmEL(nums, target):
    if target in nums:                    #如果target没出现在原表中，不写此句
        nums.remove(target)
    return nums
```

In [36]:
```
#注意不能写成，return nums.remove(target)，因为.remove()没有返回值。
# def RmEL(nums, target):
#     if target in nums:
#         return nums.remove(target)

#因为.remove()不会返回任何东西，所以必须赋值给新表，return 新表名
```

In [37]:
```
#写法2：反向：如果target【不在】新list中，【append】
def SortRm(nums):
    result = []
    for num in nums:
        if num not in result:
            result.append(num)
    return result

#因为已经是sorted list，就不需要再append后再sort了。
```

In [38]:
```
#写法3：pop1:tmp定为第一个位置元素，i指针指向第二位：如果i指针上元素与tmp相同，

def SortRm(nums):
    if len(nums) <= 1:
        return nums
    tmp = nums[0]
    i = 1
    while i < len(nums):
        if nums[i] == tmp:
            nums.pop(i)
        else:
            tmp = nums[i]
            i += 1
    return nums

#https://pythontutor.com/render.html#code=def%20SortRm%28nums%29%3A%(
```

## 3.4 Merge Two Sorted Lists

Description Build a function MergeSorted which merges two sorted lists as one sorted list.

Examples MergeSorted([1, 3, 5], [2, 4, 6]) returns [1, 2, 3, 4, 5, 6]

MergeSorted([1, 2, 3], [4, 5, 6]) returns [1, 2, 3, 4, 5, 6]

两个排好序的list合并

In [39]:
```
#写法1：直接return sorted(list1 + list2)
def MergeSorted(Lst1, Lst2):
    return sorted(Lst1 + Lst2)
```

In [40]:
```
#写法2：
def MergeSorted(nums1, nums2):
    nums3 = nums1 + nums2
    return sorted(nums3)
```

In [41]:
```
#写法3: i = 0, 如果list1[0]小于list2[i]个值, 则把list1[0] pop出来, 插入在li

def MergeSorted(Lst1, Lst2):
    i = 0
    while Lst1 and i < len(Lst2):
        if Lst1[0] <= Lst2[i]:
            num = Lst1.pop(0)
            Lst2.insert(i, num)
        i += 1

    if Lst1:
        return Lst2 + Lst1

    return Lst2
```

## 3.5 List to Number

Description Build a function Lst2Num which convert a number list into one number.

Examples Lst2Num([1, 3, 5]) returns 135

Lst2Num([0, 1, 2, 0, 3]) returns 1203

把一个序列转化为一个数字

In [42]:
```
def Lst2Num(nums):
    result = 0
    for num in nums:
        result = result * 10 + num
    return result
```

In [43]:
```
#字符串拼接:
def Lst2Num(lst):
    num_str = ''.join(map(str, lst))
    num = int(num_str)
    return num
```

## 3.6 Maximum Stock Gain

Description Build a function MaxGain to find the maximum you can gain by buying and selling stocks. The strock prices represented as a list of numbers. You need to buy stocks before you sell them.

Examples MaxGain([3, 4, 1, 5, 7, 2]) returns 6 --Explanation: Buy stocks when the price is 1 and sell them when the price is 7.

MaxGain([5, 4, 3, 2, 1]) returns 0

后值减去前值的最大值

In [44]:
```
def MaxGain(prices):
    max_gain = 0
    for i in range(0, len(prices) - 1):
        for j in range(i+1, len(prices)):
            gain = prices[j] - prices[i]
            if gain > max_gain:
                max_gain = gain
    return max_gain


#①：i in range(0, len(num) - 1)：从第一个元素，到倒数第二个元素
#②：j in range(i+1, len(num))：从i后的第一个元素，到最后一个元素
#③：每一次在i值固定时，j每一次向后移动一个，用[j]值减去[i]值，如果gain超过max_
#④：在j值走完后，让i值再向后一个，再在i值固定下走完所有j值。
#link: https://pythontutor.com/render.html#code=def%20MaxGain%28price
```

## 3.7 Sort List of Lists

Description Build a function SortLL which sorts a list of lists by their last elements.

Examples SortLL([[1, 3, 5], [2, 4], [9, 7], [3]]) returns [[3], [2, 4], [1, 3, 5], [9, 7]]

In [45]:
```
def SortLL(Lsts):
    return sorted(Lsts, key = lambda x:x[-1])

#用lsts来表示大list，x表示小list，x[-1]表示小list中最后一个值。
#sorted(xxx, key)的key参数是一个函数，用于指定排序的规则。
#lambda表达式：lambda x:x[-1]来表示每个小list的最后一个值
#最外面的sorted表示进行排序。
```

In [46]:
```
#错误答案：
# def SortLL(Lsts):
#     sorted(Lsts, key = lambda x:x[-1])
#     return Lsts

# 不能使用的原因：
# sorted() 函数会返回一个新的排序后的列表，但它不会改变原始列表。必须写一个新的l
```

In [47]:
```
#修改1：
def SortLL(Lsts):
    Lsts = sorted(Lsts, key=lambda x: x[-1])
    return Lsts
```

In [48]:
```
#等同于：
def SortLL(Lsts):
    return sorted(Lsts, key=lambda x: x[-1])
```

In [49]:
```
#修改2：
def SortLL(Lsts):
    Lsts.sort(key=lambda x: x[-1])
    return Lsts

#注：sorted和.sort的区别和使用：
#sorted返回新链表但不修改原始链表。所以要return 新起表名，或直接return sorted
#.sort()不返回新链表但修改了原始链表。所以要return 旧表名有结果。
#所以可以写return sorted(Lsts, key=lambda x: x[-1])；但不可以写return Ls
```

## 3.8 Pascals Triangle

Description Build a function PascalsT which takes an integer n as an input and generate a n layer Pascals Triangle.

Examples PascalsT(3) returns [[1], [1, 1], [1, 2, 1]]

PascalsT(5) returns [[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]

杨辉三角 每一层数字的数目等于层数 最左边和最右边的数字都是1 从第三层开始，中间第i个数字的值，是上一层第i和第i-1数字的和 里面每一个元素是一个list

In [50]:
```
def PascalsT(n):
    triangle = []
    for i in range(n):
        row = [1] * (i + 1)  # 创建当前行，初始元素都为1
        for j in range(1, i):
            row[j] = triangle[i-1][j-1] + triangle[i-1][j]  # 根据前一
        triangle.append(row)
    return triangle

# 它使用了相同的原则来生成帕斯卡三角形。我们遍历每一层，并根据帕斯卡三角形的规律计
```

# 4. String

## 4.1 All Red

Description Build a function AllRed that takes a list of strings as input, returns a boolean indicating whether all the strings containing word "red".

Examples AllRed(["red hat", "a pair of red shoes", "three red apples"]) returns True

AllRed(["red hat", "white shirt", "black eyes"]) returns False

AllRed([]) returns False

检索如果string中每个词都有red，则return True, 否则return False

In [51]:
```
def AllRed(Lst):
    if not Lst:                 #如果Lst为空
        return False              #返回false
    for s in Lst:               #对于在Lst中的每个元素s,
        if 'red' not in s:        #如果red不在其中
            return False            #返回false
    return True                 #否则返回True
```

In [52]:
```
AllRed(["red hat", "a pair of red shoes", "three red apples"])
```

Out[52]: True

In [53]:
```
AllRed(["red hat", "white shirt", "black eyes"])
```

Out[53]: False

## 4.2 Reverse String

Description Build a function RevStr to reverse a string.

Examples RevStr("abcd") returns "dcba"

把string字母序逆序

In [54]:
```
def RevStr(s):          #定义RevStr函数，接收s
    return s[::-1]      #s[::-1]把每个元素reverse
```

In [55]:
```
RevStr("abcd")
```

Out[55]: 'dcba'

## 4.3 Reverse Sentence

Description Build a function RevSen to reverse a sentence.

Examples RevSen("we are all friends") returns "friends all are we”

拆句成词，句子词序逆序，保留每个词内的字母顺序

In [56]:
```
def RevSen(sen):                    #定义RevSen函数，接收sen
    words = sen.split()             #words = sen句子拆分成词
    return " ".join(words[::-1])    #return 用空格来合并（倒序的，拆分后每
```

In [57]:
```
RevSen("we are all friends")
#'friends all are we'
```

Out[57]: 'friends all are we'

In [58]:
```
#分解打印式理解:
def RevSen(sen):
    words = sen.split()
    print("words = ", words)

    new_words = words[::-1]
    print("new_words", new_words)

    res = " ".join(new_words)
    print("res", res)

    return res
```

In [59]:
```
RevSen("we are all friends")
```

```
words =  ['we', 'are', 'all', 'friends']
new_words ['friends', 'all', 'are', 'we']
res friends all are we
```

Out[59]:
```
'friends all are we'
```

## 4.4 Reverse Words

Reverse Words Description Build a function RevWords to reverse each word of a sentence.

Examples RevWords("we are all friends") returns "ew era lla sdneirf"

拆句成词，保持句子词序，将每个词内的字母倒序

In [60]:
```
def RevWords(sen):
    words = sen.split()                          #拆句成词
    res = ""                                     #result为空
    for word in words:                           #对于在words中的word
        print("word = ",word)
        if res:                                  #如果result不为空
            res += " "                           #就在result后加个空格
            print("res with space = ", res)
        res += word[::-1]                        #将word内词序逆序，加到resu
        print("word[::-1]=", word[::-1])
        print("res = ", res)
        print("========")
    return res
```

In [61]:
```
RevWords("we are all friends")
```

```
word =  we
word[::-1]= ew
res =  ew
========
word =  are
res with space =  ew
word[::-1]= era
res =  ew era
========
word =  all
res with space =  ew era
word[::-1]= lla
res =  ew era lla
========
word =  friends
res with space =  ew era lla
word[::-1]= sdneirf
res =  ew era lla sdneirf
========
```

## 4.5 Capitalize First Letters

Description Build a function CapLett to capitalize the first letter of each word in a sentence.

Examples CapLett("we are all friends") returns "We Are All Friends"

拆句成词，保持句子词序，让每个词首字母大写

In [62]:
```
#注意缩进顺序:
def CapLett(sen):
    res = ""                          #设置result为空
    for word in sen.split():          #对于在句子拆分状态下的词:
        if res:                       #如果result不为空
            res += " "                #就把每个result中间空格
        res += word.capitalize()      #让每个词首字母大写
    return res
```

In [63]:
```
CapLett("we are all friends")
#'We Are All Friends'
```

Out[63]:
```
'We Are All Friends'
```

In [64]:
```
#拆解式分解：
def CapLett(sen):
    res = ""                          #设置result为空
    for word in sen.split():          #对于在句子拆分状态下的词：
        print("word:",word)
        if res:                       #如果result不为空
            res += " "                #就把result后加一个空格
            print("res with space:", res)
        res += word.capitalize()  #让新词首字母大写，放到result中
        print("res:",res)
    return res
```

In [65]:
```
CapLett("we are all friends")
#将句子拆分成词，检索到第一个词是we，因为result为空，所以不加空格，直接大写首字母
#第二个词是are，因为result已有We，不为空，就将其后加一个空格；让are首字母大写，
```

```
word: we
res: We
word: are
res with space: We
res: We Are
word: all
res with space: We Are
res: We Are All
word: friends
res with space: We Are All
res: We Are All Friends
```

Out[65]:
```
'We Are All Friends'
```

## 4.6 Remove Punctuations

Description Build a function RmPunct to remove punctuations from a string.

Examples RmPunct("He said, that is great!!") returns "He said that is great"

把句子中的标点去掉

In [66]:
```
import string
def RmPunct(s):                       #RmPunct函数，接收string
    res = ""                          #result为空
    punts = set(string.punctuation)   #punts设置为string的标点
    for c in s:                       #对每个在string中的character
        print("c:", c)
        if c not in punts:            #如果不在punctuation中
            res += c                  #就加到result中
            print("res:", res)
    return res
```

In [67]:
```
RmPunct("He said, 'that is great!!")
#'He said that is great'
#走到,空格 '' !! 时，就不会将其加入到result中
```

```
c: H
res: H
c: e
res: He
c:
res: He
c: s
res: He s
c: a
res: He sa
c: i
res: He sai
c: d
res: He said
c: ,
c:
res: He said
c: '
c: t
res: He said t
c: h
res: He said th
c: a
res: He said tha
c: t
res: He said that
c:
res: He said that
c: i
res: He said that i
c: s
res: He said that is
c:
res: He said that is
c: g
res: He said that is g
c: r
res: He said that is gr
c: e
res: He said that is gre
c: a
res: He said that is grea
c: t
res: He said that is great
c: !
c: !
```

Out[67]: 'He said that is great'

In [68]:
```
RmPunct("10001 He said, 'that is great!'")
#'10001 He said that is great'
```

```
c: 1
res: 1
c: 0
res: 10
c: 0
res: 100
c: 0
res: 1000
c: 1
res: 10001
c:
res: 10001
c: H
res: 10001 H
c: e
res: 10001 He
c:
res: 10001 He
c: s
res: 10001 He s
c: a
res: 10001 He sa
c: i
res: 10001 He sai
c: d
res: 10001 He said
c: ,
c:
res: 10001 He said
c: '
c: t
res: 10001 He said t
c: h
res: 10001 He said th
c: a
res: 10001 He said tha
c: t
res: 10001 He said that
c:
res: 10001 He said that
c: i
res: 10001 He said that i
c: s
res: 10001 He said that is
c:
res: 10001 He said that is
c: g
res: 10001 He said that is g
c: r
res: 10001 He said that is gr
c: e
res: 10001 He said that is gre
c: a
res: 10001 He said that is grea
c: t
res: 10001 He said that is great
c: !
c: '
```

Out[68]: '10001 He said that is great'

## 4.7 N Grams

Description Build a function NGram which takes a string s and an integer n as inputs, returns a list of n grams of s.

Examples

NGram("techlent", 2) returns ["te", "ec", "ch", "hl", "le", "en", "nt"]

NGram("abc", 3) returns ["abc"]

NGram("abc", 4) returns [ ]

NGram("abc", 1) returns ["a", "b", "c"]

在string中取n个元素为一组，取多组

In [69]:
```
def NGram(s, n):
    res = []
    print("len(s) - n + 1 = ", len(s) - n + 1)
    for i in range(len(s) - n + 1):
        print("i = ", i)
        print("i + n = ", i + n)
        print(s[i: i + n])
        print("===")
        res.append(s[i: i + n])
    return res

#s代表字符串，n是需要几个字符:
#lens(s) - n + 1的作用,
#是i运行到 元素总数 - n 个元素时停止
```

In [70]:
```
NGram("techlent", 2)
```

```
len(s) - n + 1 =  7
i =  0
i + n =  2
te
===
i =  1
i + n =  3
ec
===
i =  2
i + n =  4
ch
===
i =  3
i + n =  5
hl
===
i =  4
i + n =  6
le
===
i =  5
i + n =  7
en
===
i =  6
i + n =  8
nt
===
```

Out[70]:
```
['te', 'ec', 'ch', 'hl', 'le', 'en', 'nt']
```

# 4.8 Palindrome

Description Build a function isPalindrome which takes a string as an input and returns boolin value telling whether the string is a palindrome.

Examples

isPalindrome("abcba") returns True

isPalindrome("abba") returns True

isPalindrome("a") returns True

isPalindrome("") returns True

isPalindrome("abcd") returns False

如果所有元素反过来还相等，则返回True

In [71]:
```
#法1：s = s[::-1]让所有元素逆序
def isPalindrome(s):
    return s == s[::-1]
```

In [72]:
```
isPalindrome("abcba")
```

Out[72]: True

In [73]:
```
isPalindrome("abcd")
```

Out[73]: False

In [74]:
```
#法2：指针法
def isPalindrome(s):
    left = 0
    right = len(s) - 1
    while left < right:                    #走到最后，奇数个时，s[left] = s
        print("left = ", left)
        print("right = ", right)
        print("s[left] = ", s[left])
        print("s[right] = ", s[right])
        print("====")
        if s[left] == s[right]:
            left += 1
            right -= 1
        else:
            return False
    return True
```

In [75]:
```
isPalindrome("abba")
```

```
left =  0
right =  3
s[left] =  a
s[right] =  a
====
left =  1
right =  2
s[left] =  b
s[right] =  b
====
```

Out[75]: True

In [76]:
```
isPalindrome("ababa")
```

```
left =  0
right =  4
s[left] =  a
s[right] =  a
====
left =  1
right =  3
s[left] =  b
s[right] =  b
====
```

Out[76]:
```
True
```

In [77]:
```
isPalindrome("abcd")
```

```
left =  0
right =  3
s[left] =  a
s[right] =  d
====
```

Out[77]:
```
False
```

## 4.9 Letter Palindrome

Description Build a function LettPalind to determine a string is a palindrome or not. We only consider letters this time.

Examples

LettPalind("909@,.") returns True

LettPalind("He is Sieh!") returns True

LettPalind("His name is Sieh.") returns False

判断一个句子是否是回文

In [78]:
```
import string
def LettPalind(s):
    letts = set(string.ascii_lowercase)
    s = s.lower()
    left = 0
    right = len(s) - 1
    while left < right:
        print("left = ", left)
        print("right = ", right)
        print("s[left] = ", s[left])
        print("s[right] = ", s[right])
        if s[left] not in letts:
            left += 1
        elif s[right] not in letts:
            right -= 1
        elif s[left] == s[right]:
            left += 1
            right -= 1
        else:
            return False
    return True

#当遇到左手或右手不是字母时，就让左手后进一位，或右手前移一位
#当左右手相等时，各自后进/前移一位，直到打破循环
#如果左右手不相等时，返回false
```

In [79]:
```
LettPalind("He is Sieh!")
```

```
left =  0
right =  10
s[left] =  h
s[right] =  !
left =  0
right =  9
s[left] =  h
s[right] =  h
left =  1
right =  8
s[left] =  e
s[right] =  e
left =  2
right =  7
s[left] =
s[right] =  i
left =  3
right =  7
s[left] =  i
s[right] =  i
left =  4
right =  6
s[left] =  s
s[right] =  s
```

Out[79]: True

In [80]:
```
LettPalind("His name is Sieh!")
```

```
left =  0
right =  16
s[left] =  h
s[right] =  !
left =  0
right =  15
s[left] =  h
s[right] =  h
left =  1
right =  14
s[left] =  i
s[right] =  e
```

Out[80]:
```
False
```

## 4.10 Valid Parenthesis

Description Build a function ValidParenthesis to determine the parenthesis in a string are valid.

Examples

ValidParenthesis("((()))()") returns True

ValidParenthesis("()()()") returns True

ValidParenthesis("()()(") returns False

ValidParenthesis("((()))") returns False

如果左右括弧数相等，则返回True，否则False

In [81]:
```
def ValidParenthesis(s):
    tmp = []
    for c in s:
        print("tmp=", tmp)
        print("c=", c)
        print("=================")
        if c == "(":
            tmp.append(c)
        if c == ")":
            if tmp:
                tmp.pop()
            else:
                return False
    print("final tmp=", tmp)
    if tmp:
        return False
    else:
        return True

#遇到元素是左括弧 (，存入tmp，遇到元素是右括弧)，弹出
#如果右括弧弹无可弹，return False： 左括弧数量 > 右括弧数量
#如果走完全程，tmp还有值：右括弧数量 > 左括弧数量
#如果走完全程，tmp没有值：两个括号数量 =
```

In [82]:
```
#退无可退，return False
ValidParenthesis('((()))()')
```

```
tmp= []
c= (
=================
tmp= ['(']
c= (
=================
tmp= ['(', '(']
c= )
=================
tmp= ['(']
c= )
=================
tmp= []
c= )
=================
```

Out[82]: False

In [83]:
```
#走完全程，tmp仍遗留元素，return False
ValidParenthesis('(((()))()')
```

```
tmp= []
c= (
==================
tmp= ['(']
c= (
==================
tmp= ['(', '(']
c= (
==================
tmp= ['(', '(', '(']
c= (
==================
tmp= ['(', '(', '(', '(']
c= )
==================
tmp= ['(', '(', '(']
c= )
==================
tmp= ['(', '(']
c= )
==================
tmp= ['(']
c= (
==================
tmp= ['(', '(']
c= )
==================
final tmp= ['(']
```

Out[83]: False

In [84]:
```
#走完全程，tmp无值，return True
ValidParenthesis('((()))()')
```

```
tmp= []
c= (
==================
tmp= ['(']
c= (
==================
tmp= ['(', '(']
c= (
==================
tmp= ['(', '(', '(']
c= )
==================
tmp= ['(', '(']
c= )
==================
tmp= ['(']
c= )
==================
tmp= []
```

# 5. Dictionary

## 5.1 Key Value Switch

Description Build a function kvSwitch to switch the keys and values of a dictionary.

Examples kvSwitch({"a": "1", "b": "2", "c": "3"}) returns {"1": "a", "2": "b", "3": “c"}

把key和value调换

In [85]:
```
#法1：
def kvSwitch(d):
    print(d.items())
    return {v:k for k,v in d.items()}

#创建kvSwitch函数，接收dictionary
#对于key, value pair, 返回value:key
```

In [86]:
```
kvSwitch({"a":1, "b":2, "c":3})
```

```
dict_items([('a', 1), ('b', 2), ('c', 3)])
```

Out[86]:
```
{1: 'a', 2: 'b', 3: 'c'}
```

In [87]:
```
#法2：
def kvSwitch(d):
    new_dict = {}
    for k,v in d.items():
        new_dict[v] = k
        print("k = ", k)
        print("v = ", v)
        print("new_dict = ", new_dict)
        print("===")
    return new_dict

#创建kvSwitch函数，接收dictionary
#设新字典为空；对于字典中的key, value pair, 让value = key；返回新字典
```

In [88]:
```
kvSwitch({"a": 1, "b": 2, "c": 3})
```

```
k =  a
v =  1
new_dict =  {1: 'a'}
===
k =  b
v =  2
new_dict =  {1: 'a', 2: 'b'}
===
k =  c
v =  3
new_dict =  {1: 'a', 2: 'b', 3: 'c'}
===
```

Out[88]:
```
{1: 'a', 2: 'b', 3: 'c'}
```

## 5.2 Rank Keys by Values

Description Build a function RankK which takes a dictionary as input returns a list of keys which sorted by their values.

Examples RankK({"a": 7, "b": 5, "c": 9}) returns ["b", "a", "c"]

根据字典的value，给key排序 eg. RF中Feature Importance；NLP重复次数找关键词

In [89]:
```
#写法1：
def RankK(d):
    items = sorted(d.items(), key = lambda x:x[-1])
    res = []
    for item in items:
        res.append(item[0])
    return res

#设置rankK函数，接收dictionary
#对于d.items，根据value进行排序，命名为items
#设置result list为空
#对于在items中的item，将第一个值（即key），append到result中，返回result
```

In [90]:
```
RankK({"a": 7, "b": 5, "c": 9})
```

Out[90]:
```
['b', 'a', 'c']
```

In [91]:
```
#写法2：
def RankK(d):
    items = d.items()
    print("items = ", items)
    sorted_items = sorted(items, key = lambda x:x[-1])
    print("sorted_items = ", sorted_items)
    res = []
    for item in sorted_items:
        res.append(item[0])
    return res

# 设置rankK函数，接收dictionary
# 对于d.items()，根据value的大小，进行排列，命名为sorted_items
# 设置result为空集，
# 对于sorted_items中的item，取第一个值（即key），返回result list
```

In [92]:
```
RankK({"a": 7, "b": 5, "c": 9})
```

```
items =  dict_items([('a', 7), ('b', 5), ('c', 9)])
sorted_items =  [('b', 5), ('a', 7), ('c', 9)]
```

Out[92]:
```
['b', 'a', 'c']
```

## 5.3 Character Frequency

Description Build a function ChaFreq to calculate the frequencies of characters in a string. Store the results in a dictionary.

Examples ChaFreq("techlent") returns {"c": 1, "e": 2, "h": 1, "l": 1, "n": 1, "t": 2}

计算字符出现频率

In [93]:
```
def ChaFreq(s):
    res = {}
    for c in s:
        print("c = ", c)
        if c in res:
            res[c] += 1
        else:
            res[c] = 1
        print("res", res)
    return res

# 设置ChaFreq函数，接收string
# 设置result为空集
# 对于string中的character，如果character在result中，则给character的value
# 如果character没在result中，则把character放到dict中，并将其value赋为1
# 返回result list
```

In [94]:
```
ChaFreq("techlent")
```

```
c =  t
res {'t': 1}
c =  e
res {'t': 1, 'e': 1}
c =  c
res {'t': 1, 'e': 1, 'c': 1}
c =  h
res {'t': 1, 'e': 1, 'c': 1, 'h': 1}
c =  l
res {'t': 1, 'e': 1, 'c': 1, 'h': 1, 'l': 1}
c =  e
res {'t': 1, 'e': 2, 'c': 1, 'h': 1, 'l': 1}
c =  n
res {'t': 1, 'e': 2, 'c': 1, 'h': 1, 'l': 1, 'n': 1}
c =  t
res {'t': 2, 'e': 2, 'c': 1, 'h': 1, 'l': 1, 'n': 1}
```

Out[94]:
```
{'t': 2, 'e': 2, 'c': 1, 'h': 1, 'l': 1, 'n': 1}
```

## 5.4 Unique Only

Description Build a function UniqueOnly to detect whether a string only contains unique characters.

Examples UniqueOnly("techlent") returns False UniqueOnly("abcde") returns True

看是否只包含唯一的字符

In [95]:
```
#法1：数frequency

def UniqueOnly(s):
    tmp = {}
    for c in s:
        print("c = ", c)
        if c in tmp:
            return False
        else:
            tmp[c] = 1
        print("tmp = ", tmp)
    return True

#设置UniqueOnly函数，接收string
#设tmp为空集
#对于在string中的character，如果character在tmp中，return False，
#如果不在，则将其放到tmp dictionary中，value赋为1
#如果所有character都走完，结束循环，return True
```

In [96]:
```
UniqueOnly("techlent")
```

```
c =  t
tmp =  {'t': 1}
c =  e
tmp =  {'t': 1, 'e': 1}
c =  c
tmp =  {'t': 1, 'e': 1, 'c': 1}
c =  h
tmp =  {'t': 1, 'e': 1, 'c': 1, 'h': 1}
c =  l
tmp =  {'t': 1, 'e': 1, 'c': 1, 'h': 1, 'l': 1}
c =  e
```

Out[96]:
```
False
```

In [97]:
```
UniqueOnly("abcde")
```

```
c =  a
tmp =  {'a': 1}
c =  b
tmp =  {'a': 1, 'b': 1}
c =  c
tmp =  {'a': 1, 'b': 1, 'c': 1}
c =  d
tmp =  {'a': 1, 'b': 1, 'c': 1, 'd': 1}
c =  e
tmp =  {'a': 1, 'b': 1, 'c': 1, 'd': 1, 'e': 1}
```

Out[97]:
```
True
```

In [98]:
```
#法2：set长度法

def UniqueOnly(s):
    return len(s) == len(set(s))

# 理论：set是没有value的dictionary，如有重复值，则将其自动去掉
# 如果string长度 = set长度，说明字符都是unique

# 设置UniqueOnly函数，接收string
# 如果string长度 = set(string)长度，则说明字符唯一，返回True，否则False
```

In [99]:
```
UniqueOnly("techlent")
```

Out[99]:
```
False
```

In [100]:
```
UniqueOnly("abcde")
```

Out[100]:
```
True
```

## 5.5 Same Pattern（难）

Description Build a function SamePattern takes two strings as inputs and determines whether they are in the same pattern.

Examples SamePattern("egg", "zoo") returns True SamePattern("abb", "ab") returns False SamePattern("hello", "agree") returns False SamePattern("aaa", "abc") returns False SamePattern("abc", "aaa") returns False

判断两个string是否模式相同：

本质上就是如果当前字母没有被储存过value， 就将string1[i]:string2[i]，放进dict中做成key value pair

如果被储存过，走到当前字母时，查看储存的value和实际value是否相等

如果等、则走下去；否则return False

In [101]:
```
def SamePattern(s1, s2):
    if len(s1) != len(s2):
        return False
    d1 = {}
    d2 = {}
    for i in range(len(s1)):
        print("===")
        print("i = ", i)
        print("s1[i] = ", s1[i])
        print("s2[i] = ", s2[i])
        print("d1 = ", d1)
        print("d2 = ", d2)
        if s1[i] in d1:
            if d1[s1[i]] != s2[i]:
                return False
        else:
            d1[s1[i]] = s2[i]

        if s2[i] in d2:
            if d2[s2[i]] != s1[i]:
                return False
        else:
            d2[s2[i]] = s1[i]
        print("new_d1=", d1)
        print("new_d2=", d2)
    return True
```

In [102]:
```
SamePattern('egg', 'zoo')
#当i=0时，s1[0] = e, s2[0] = z,
#因为d1为空，则d1[e] = s2[0] = z, 又因d2为空，d2[z] = s1[0] = e
#即d1 = {'e':'z'}, d2 = {'z':'e'}

#当i=1时，s1[1] = g, s2[1] = o,
# 因为s1[1] = g没在d1中，则d1[g] = o, 又因为s2[1] = o没在d2中，d2[o] = g
# 即d1 = {'e':'z', 'g':'o'}, d2 = {'z':'e', 'o':'g'}

# 当i = 2时，s1[2] = g, 在d1中，且d1[g] = s2[2] = o，则不return False，走
# 即d1= {'e': 'z', 'g': 'o'}
# 当i = 2时，s2[2] = o, 在d2中，且d2[o] = s1[2] = g，则不return False，走
# 即d2= {'z': 'e', 'o': 'g'}

#i = 2:
#此时s1[i]=g在d1中，看在d1中g所对应的字符，是否是现在的s2[i]即o；因为相等，所以
#此时s2[i]=o在d2中，看在d2中o对应的字符，是否是现在的s1[i]即g；因为相等，所以不
```

```
new_d1= {'e': 'z'}
new_d2= {'z': 'e'}
===
i =  1
s1[i] =  g
s2[i] =  o
d1 =  {'e': 'z'}
d2 =  {'z': 'e'}
new_d1= {'e': 'z', 'g': 'o'}
new_d2= {'z': 'e', 'o': 'g'}
===
i =  2
s1[i] =  g
s2[i] =  o
d1 =  {'e': 'z', 'g': 'o'}
d2 =  {'z': 'e', 'o': 'g'}
new_d1= {'e': 'z', 'g': 'o'}
new_d2= {'z': 'e', 'o': 'g'}
```

Out[102]:
```
True
```

In [103]:
```
SamePattern('egt', 'zoo')

# d1= {'e': 'z', 'g': 'o'}
# d2= {'z': 'e', 'o': 'g'}
# 当i = 2时, s1[2] = t, 不在d1中
# 当i = 2时, s2[2] = o, 在d2中, 但d2[o] = g, 与s1[2] = t 不等,  则return

# key是string2中的字母, value和实际值是string1中的对应字母
# 当走到dictionary中的key时, 如果储存的value与实际值不等, 则return False
```

```
===
i =  0
s1[i] =  e
s2[i] =  z
d1 =  {}
d2 =  {}
new_d1= {'e': 'z'}
new_d2= {'z': 'e'}
===
i =  1
s1[i] =  g
s2[i] =  o
d1 =  {'e': 'z'}
d2 =  {'z': 'e'}
new_d1= {'e': 'z', 'g': 'o'}
new_d2= {'z': 'e', 'o': 'g'}
===
i =  2
s1[i] =  t
s2[i] =  o
d1 =  {'e': 'z', 'g': 'o'}
d2 =  {'z': 'e', 'o': 'g'}
```

Out[103]:
```
False
```

## 5.6 Anagrams

Description Build a function anagrams to determine whether two given strings are anagrams.

Examples anagrams("abc", "cba") returns True

anagrams("ab", "abc") returns False

anagrams("abccba", "aabbbc") returns False

字符出现的频率相同, 则返回True

In [104]:
```
def anagrams(s1, s2):
    d1 = {}
    for c in s1:
        print("c = ", c)
        if c in d1:
            d1[c] += 1
        else:
            d1[c] = 1
        print("d1 = ", d1)
    print("==")
    for c in s2:
        print("c = ", c)
        if c in d1 and d1[c] > 0:
            d1[c] -= 1
        else:
            return False
        print("d1 = ", d1)
    return True

# 创建anagrams函数，接收string1, string2
# 设d1为空dictionary
# 对于在string1中的character, 如果character在dictionary中，则将value加1
# 如果没在，则将其放入dict中，赋值为1
# 走完所有string1的character后，开始走string2中的character
# 对于在string2中的character，如果其在dict中且value为正，则将value减一
# 否则，return False
# 如果走完string1和string2都ok，dictionary value刚好减完，则return True

#如果走到string2时，有key无法在dict中找到，说明有不同字母出现，return false
#如果走到string2时，有key的value减无可减，说明字母出现频率不同，return false
```

In [105]:
```
anagrams('abccba', 'aabbcc')
```

```
c =  a
d1 =  {'a': 1}
c =  b
d1 =  {'a': 1, 'b': 1}
c =  c
d1 =  {'a': 1, 'b': 1, 'c': 1}
c =  c
d1 =  {'a': 1, 'b': 1, 'c': 2}
c =  b
d1 =  {'a': 1, 'b': 2, 'c': 2}
c =  a
d1 =  {'a': 2, 'b': 2, 'c': 2}
==
c =  a
d1 =  {'a': 1, 'b': 2, 'c': 2}
c =  a
d1 =  {'a': 0, 'b': 2, 'c': 2}
c =  b
d1 =  {'a': 0, 'b': 1, 'c': 2}
c =  b
d1 =  {'a': 0, 'b': 0, 'c': 2}
c =  c
d1 =  {'a': 0, 'b': 0, 'c': 1}
c =  c
d1 =  {'a': 0, 'b': 0, 'c': 0}
```

Out[105]:
```
True
```

In [106]:
```
anagrams('abccba', 'aabbcce')
#当e在s2，但没在s1中，无法减时，就报错了
```

```
c =  a
d1 =  {'a': 1}
c =  b
d1 =  {'a': 1, 'b': 1}
c =  c
d1 =  {'a': 1, 'b': 1, 'c': 1}
c =  c
d1 =  {'a': 1, 'b': 1, 'c': 2}
c =  b
d1 =  {'a': 1, 'b': 2, 'c': 2}
c =  a
d1 =  {'a': 2, 'b': 2, 'c': 2}
==
c =  a
d1 =  {'a': 1, 'b': 2, 'c': 2}
c =  a
d1 =  {'a': 0, 'b': 2, 'c': 2}
c =  b
d1 =  {'a': 0, 'b': 1, 'c': 2}
c =  b
d1 =  {'a': 0, 'b': 0, 'c': 2}
c =  c
d1 =  {'a': 0, 'b': 0, 'c': 1}
c =  c
d1 =  {'a': 0, 'b': 0, 'c': 0}
c =  e
```

Out[106]: False

In [107]:
```
anagrams('abccba', 'aabbcca')
#当读取到s2最后一个a时，因为前面都减完没有可减的了，所以报错
```

```
c =  a
d1 =  {'a': 1}
c =  b
d1 =  {'a': 1, 'b': 1}
c =  c
d1 =  {'a': 1, 'b': 1, 'c': 1}
c =  c
d1 =  {'a': 1, 'b': 1, 'c': 2}
c =  b
d1 =  {'a': 1, 'b': 2, 'c': 2}
c =  a
d1 =  {'a': 2, 'b': 2, 'c': 2}
==
c =  a
d1 =  {'a': 1, 'b': 2, 'c': 2}
c =  a
d1 =  {'a': 0, 'b': 2, 'c': 2}
c =  b
d1 =  {'a': 0, 'b': 1, 'c': 2}
c =  b
d1 =  {'a': 0, 'b': 0, 'c': 2}
c =  c
d1 =  {'a': 0, 'b': 0, 'c': 1}
c =  c
d1 =  {'a': 0, 'b': 0, 'c': 0}
c =  a
```

Out[107]:
```
False
```

In [108]:
```
#法2：当sorted(s1) = sorted(s2)时，则两个string，字符出现频率相同
def anagrams(s1, s2):
    print(sorted(s1))
    print(sorted(s2))
    return sorted(s1) == sorted(s2)
```

In [109]:
```
anagrams('abc', 'cba')
```

```
['a', 'b', 'c']
['a', 'b', 'c']
```

Out[109]:
```
True
```

In [110]:
```
anagrams('ab', 'abc')
```

```
['a', 'b']
['a', 'b', 'c']
```

Out[110]:
```
False
```

## 5.7 Target Sum

Description Build a function TargetSum which takes a list and a target number as inputs. If there are two numbers in the list whose sum equals to the target number, the function returns the indice of the two numbers in list, otherwise returns -1.

Examples TargetSum([1, 2, 5, 9, 8, 11], 6) returns [0, 2] (1+5=6)

TargetSum([1, 2, 5, 9, 8, 11], 11) returns [1, 3]

TargetSum([1, 2, 5, 9, 8, 11], 5) returns -1

TargetSum([1, 2, 5, 9, 8, 11], 14) returns [2, 3]

In [111]:
```
#法1：暴力解法，时间复杂度 n * n = n^2
def TargetSum(nums, target):
    for i in range(len(nums)):
        for j in range(i):
            print("i = ", i)
            print("j = ", j)
            if (nums[i] + nums[j]) == target:
                return j, i
    return -1
```

In [112]:
```
TargetSum([1, 2, 5, 9, 8, 11], 6)
```

```
i =  1
j =  0
i =  2
j =  0
```

Out[112]:
```
(0, 2)
```

In [113]:
```
#法2：
def TargetSum(nums, target):
    tmp = {}
    for i in range(len(nums)):
        print("tmp", tmp)
        if nums[i] in tmp:
            return tmp[nums[i]], i
        else:
            tmp[target - nums[i]] = i
    return -1
```

In [114]:
```
TargetSum([1, 2, 5, 9, 8, 11], 11)
#当target value - nums[i], 放入index中
#当跑到第三个元素时，5已经在dict中了，因为之前6-1=5
#那么就把存入的5的index0，和现在5的index2拿出来即可
```

```
tmp {}
tmp {10: 0}
tmp {10: 0, 9: 1}
tmp {10: 0, 9: 1, 6: 2}
```

Out[114]:
```
(1, 3)
```